НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
"___" _____ 2026 р.	**"___" _____ 2026 р.**

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему " Програмне забезпечення формування розкладу занять для студентів за допомогою Telegram-бота "

Спеціальність 121 – «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
асистент _____ **Тетяна БАРАНОВА**
(підпис)

Консультант бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ **Ірина БОРОДКІНА**
(підпис)

Виконав _____ **Владислав ЛЯХОВЧУК**
(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ

"___" _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Ляховчуку Владиславу Вікторовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
" Програмне забезпечення формування розкладу занять для студентів за допомогою Telegram-бота "

затверджена наказом ректора від "19" грудня 2025р. № 3204 "С"

Термін подання завершеної роботи на кафедру " 22 " травня 2026р.

Вихідні дані до бакалаврської кваліфікаційної роботи:
Навчальні матеріали з розробки Telegram-ботів, баз даних та обробки розкладу
Перелік питань, що підлягають дослідженню:
Аналіз предметної області автоматизації доступу до розкладу занять студентів
Розробка програмного забезпечення для автоматизації персоналізованого розкладу занять студентів на базі Telegram-бота
Перелік графічних документів (за необхідності).

_____

Дата видачі завдання " 19 " грудня 2025р.

Керівник бакалаврської _____ **Тетяна БАРАНОВА**
кваліфікаційної роботи (підпис)

Консультант бакалаврської _____ **Ірина БОРОДКІНА**
кваліфікаційної роботи (підпис)

Завдання взяв до виконання _____ **Владислав ЛЯХОВЧУК**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 4
ВСТУП 5
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8
1.1. Опис предметної області автоматизації доступу до розкладу занять студентів 8
1.2. Аналіз вимог до програмної системи автоматизації розкладу занять студентів 11
1.3. Моделювання предметної області 13
1.4. Огляд існуючих рішень для автоматизації доступу до розкладу занять студентів 18
1.5. Постановка завдання 21
1.6. Висновки до розділу 22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ 23
2.1. Логічна модель даних у вигляді ER-діаграми 23
2.2. Вибір системи управління базами даних 26
2.3. Розробка структури інформаційної бази 28
2.4. Схема та фізична структура бази даних 31
РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РОЗКЛАДУ ЗАНЯТЬ СТУДЕНТІВ 34
3.1. Моделювання структури та взаємодії об'єктів 34
3.2. Організаційна структура програмного забезпечення 39
3.3. Компонентна структура системи 41
3.4. Вибір інструментарію для створення прикладного програмного забезпечення 45
3.5 . Алгоритмізація та програмування програмних модулів 48
РОЗДІЛ 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 52
4.1. Тестування системи 52
4.2. Вимоги до апаратного та програмного забезпечення 57
4.3. Склад інсталяційного пакету 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63
ДОДАТОК А 65
ДОДАТОК Б 92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API — програмний інтерфейс додатку (Application Programming Interface).

БД — база даних.

БОТ — програмний агент, що взаємодіє з користувачем через месенджер Telegram.

Excel — формат електронних таблиць, що використовується для збереження розкладу.

FSM — скінченний автомат станів (Finite State Machine).

HTTP — протокол передачі даних у мережі Інтернет.

HTTPS — захищений протокол передачі гіпертексту.

JSON — текстовий формат обміну даними (JavaScript Object Notation).

ORM — технологія об’єктно-реляційного відображення (Object-Relational Mapping).

Python — мова програмування, використана для реалізації програмного продукту.

SQLite — вбудована реляційна система управління базами даних.

SQL — мова структурованих запитів.

Telegram Bot API — програмний інтерфейс для створення та керування Telegram-ботами.

UML — уніфікована мова моделювання.

URL — уніфікований локатор ресурсу (Uniform Resource Locator).

UI — інтерфейс користувача (User Interface).

XLSX — формат файлів електронних таблиць Microsoft Excel.

ВСТУП

У сучасному освітньому середовищі студенти щоденно користуються цифровими засобами для отримання навчальної інформації, комунікації та організації власного часу. Одним із найважливіших елементів навчального процесу є розклад занять, оскільки саме він визначає послідовність навчальної діяльності студента, допомагає планувати день і своєчасно відвідувати заняття. Проте перегляд розкладу через вебсайти або електронні таблиці не завжди є достатньо зручним, особливо в умовах мобільного використання та необхідності швидко отримувати актуальну інформацію. [14]

Окремою проблемою є те, що стандартний розклад академічної групи не завжди повністю відповідає індивідуальному навантаженню конкретного студента. Це особливо актуально у випадках, коли в межах однієї групи студенти мають різні вибіркові дисципліни. У такій ситуації загальний розклад містить надлишкову інформацію, через що студент змушений самостійно визначати, які саме заняття належать саме йому. Це знижує зручність користування розкладом і підвищує ймовірність помилок.

Використання Telegram-бота як засобу доступу до персоналізованого розкладу дозволяє вирішити ці проблеми. Telegram є поширеною платформою, яка підтримується на смартфонах, планшетах і персональних комп’ютерах, не потребує створення окремого застосунку та забезпечує зручну взаємодію через повідомлення і меню. Бот може надавати швидкий доступ до розкладу, враховувати вибіркові дисципліни користувача, автоматично визначати тип навчального тижня та додатково реалізовувати функцію нагадувань про заняття. [3; 23]

Актуальність теми полягає в тому, що під час підготовки фахівця з інженерії програмного забезпечення важливо навчитися створювати сучасні програмні системи, які поєднують інтерфейс користувача, серверну логіку, роботу з базою даних та інтеграцію із зовнішніми джерелами інформації. Telegram-бот для автоматизації розкладу занять є прикладом практично

корисного програмного продукту, який може застосовуватися в реальному освітньому середовищі.

Мета роботи — спроєктувати й реалізувати програмне забезпечення для автоматизації доступу до персоналізованого розкладу занять студентів у вигляді Telegram-бота, який забезпечує перегляд розкладу, роботу з вибірковими дисциплінами, автоматичне визначення типу навчального тижня та підтримку нагадувань.

Завдання роботи:

1. Розглянути предметну область і вимоги до системи, описати основні сценарії роботи користувачів і структуру програмних компонентів.
2. Спроєктувати загальну архітектуру рішення та структуру бази даних, обґрунтувати вибір технологій.
3. Реалізувати програмне забезпечення: Telegram-бот, модулі роботи з розкладом, модулі роботи з вибірковими дисциплінами, підключення до бази даних, імпорт офіційного розкладу та інтерфейс користувача.
4. Реалізувати логіку автоматичного визначення типу навчального тижня та механізм персоналізації розкладу.
5. Розробити функцію нагадувань та додаткові засоби доступу до навчальної інформації.
6. Перевірити роботу основних функцій системи за тестовими сценаріями та узагальнити результати.
7. Запропонувати напрями подальшого розвитку програмного продукту.

Об’єкт дослідження — процес автоматизації доступу студентів до персоналізованого розкладу занять.

Предмет дослідження — програмне забезпечення для формування та відображення персоналізованого розкладу занять студентів на базі Telegram-бота, методи його проєктування та реалізації.

Методи дослідження: аналіз предметної області та існуючих рішень; моделювання структури системи; проєктування архітектури бази даних та

інтерфейсу користувача; програмування; тестування; узагальнення отриманих результатів. [14–17]

Наукова новизна полягає у поєднанні в одному програмному рішенні механізмів імпорту офіційного розкладу, автоматичного визначення типу навчального тижня, персоналізації занять на основі вибіркових дисциплін і реалізації доступу до цієї інформації через Telegram-бота.

Практична цінність полягає в тому, що розроблений програмний продукт може бути використаний студентами для щоденного перегляду актуального персоналізованого розкладу занять, а також може слугувати основою для подальшого розвитку подібних інформаційних систем в освітньому середовищі.

Структура записки: вступ; розділ з аналізом предметної області та вимог; розділ з проєктуванням архітектури та бази даних; розділ з описом програмної реалізації; розділ з тестуванням і вимогами до впровадження; висновки; список джерел; додатки з діаграмами, фрагментами коду та матеріалами перевірки.

Таким чином, робота спрямована на створення практично корисного програмного продукту, який забезпечує автоматизацію доступу до розкладу занять студентів і демонструє повний цикл розробки сучасної інформаційної системи на базі Telegram-бота.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області автоматизації доступу до розкладу занять студентів

Організація навчального процесу у закладах вищої освіти тісно пов’язана з необхідністю швидкого доступу студентів до актуального розкладу занять. Розклад є базовим інформаційним ресурсом, який визначає послідовність навчальних подій, допомагає планувати день, орієнтуватися у змінах навчального процесу та своєчасно відвідувати заняття. У сучасних умовах цифровізації освіти користувачі очікують отримувати цю інформацію оперативно, у зручному форматі та з будь-якого доступного пристрою. [14]

Практика показує, що традиційні способи подання розкладу, зокрема через вебсторінки університетів або електронні таблиці, не завжди є достатньо зручними. Такі рішення часто орієнтовані на відображення загального розкладу для всіх груп або потоків, а не на індивідуальні потреби конкретного студента. У результаті користувач змушений самостійно знаходити потрібну групу, визначати актуальний день, тип навчального тижня та відокремлювати власні заняття від зайвої інформації.

Особливістю предметної області є те, що в межах однієї академічної групи студенти можуть мати різні вибіркові дисципліни. Це означає, що навіть за наявності спільного базового розкладу фактичний набір занять для різних студентів відрізняється. Таким чином, предметна область охоплює не лише задачу відображення загального розкладу, а й задачу його персоналізації відповідно до індивідуального навчального плану користувача.

У поняттєвому плані предметну область доцільно описувати через такі основні сутності: користувач, профіль користувача, розклад, навчальне заняття, вибіркова дисципліна, тип навчального тижня, нагадування та джерело даних розкладу.

Користувач системи — це студент, який взаємодіє з програмним продуктом для отримання інформації про свої заняття. Профіль користувача

містить індивідуальні параметри, зокрема вибрані дисципліни та стан нагадувань. Навчальне заняття є окремим елементом розкладу, що має назву дисципліни, день проведення, номер пари, тип тижня та, за потреби, додаткову інформацію. Вибіркова дисципліна є окремим видом навчального предмета, що обирається студентом і повинна враховуватися при формуванні персоналізованого розкладу.

Окреме значення у предметній області має тип навчального тижня. У багатьох закладах вищої освіти розклад будується за принципом чергування чисельника і знаменника, через що одні й ті самі часові слоти можуть містити різні заняття залежно від поточного тижня. Тому для коректної роботи системи потрібно не лише зберігати розклад, а й визначати актуальний тип тижня автоматично.

Ще одним важливим аспектом є структура самих джерел даних. На практиці розклад часто публікується у вигляді електронних таблиць Excel або генерується на основі офіційних даних університету. Це означає, що система повинна вміти працювати з напівструктурованими даними, зчитувати записи з таблиць, знаходити потрібну академічну групу, інтерпретувати об’єднані комірки, розрізняти типи занять та переносити ці дані у внутрішню структуру програмного забезпечення. [5; 6]

У сучасному інформаційному середовищі важливим є також вибір каналу взаємодії з користувачем. Для студентської аудиторії доцільним є використання месенджера Telegram, оскільки він є звичним, доступним на різних платформах і дозволяє організувати простий діалоговий інтерфейс без потреби створення окремого мобільного або вебзастосунку. Telegram-бот у такій предметній області виступає як програмний інтерфейс доступу до розкладу, що поєднує меню, повідомлення, кнопки та логіку персоналізації. [3; 23]

Предметна область даної роботи охоплює галузь програмних засобів автоматизації доступу до розкладу занять студентів із використанням Telegram-бота як інтерфейсу взаємодії. Вона поєднує питання організації навчального процесу, роботи з офіційними джерелами даних, персоналізації відображення

занять, визначення типу навчального тижня, збереження користувацьких налаштувань та забезпечення зручного щоденного доступу до навчальної інформації. Подальший аналіз сучасних підходів і програмних засобів у цій сфері дозволяє обґрунтувати вимоги до конкретної системи та вибір технологій для її реалізації в наступних розділах записки.

1.2. Аналіз вимог до програмної системи автоматизації розкладу занять студентів

Вимоги до системи автоматизації доступу до розкладу занять зводяться до того, щоб користувач міг швидко отримати актуальну інформацію про свої заняття, не витрачаючи час на пошук потрібної групи, перевірку типу навчального тижня та відокремлення власних дисциплін від загального розкладу. Система повинна бути простою у використанні, доступною з мобільного пристрою та не вимагати встановлення окремого програмного забезпечення, окрім звичного для користувача месенджера Telegram. Окремо важливо врахувати коректність обробки вибіркових дисциплін, автоматичне визначення типу навчального тижня та зручність інтерфейсу, щоб студент міг швидко зорієнтуватися в роботі бота. [14]

З функціональної точки зору система повинна забезпечувати запуск Telegram-бота, відображення головного меню та доступ до основних функцій. Користувач повинен мати можливість переглядати розклад на сьогодні, завтра та поточний тиждень. Бот має враховувати індивідуально обрані вибіркові дисципліни користувача та формувати персоналізований розклад. Також система повинна дозволяти переглядати список власних вибіркових дисциплін, змінювати їх, переглядати профіль користувача та працювати з нагадуваннями про заняття. Для повноцінної роботи програмного продукту має бути реалізовано імпорт офіційного розкладу з Excel-файлу, збереження даних у базі та подальше використання цих даних під час формування відповіді користувачу.

Окремою функціональною вимогою є визначення типу заняття. Система повинна розрізняти лекційні та практичні заняття на основі структури вхідного Excel-файлу. Якщо запис про дисципліну є спільним для кількох академічних груп, його доцільно трактувати як лекційне заняття. Якщо ж заняття подане окремо для кожної групи, воно має визначатися як практичне. Такий підхід дозволяє зробити розклад більш інформативним і в подальшому використовувати цю ознаку для розширення функціоналу, зокрема додавання посилань на Google Meet для лекційних занять. [5; 6]

З нефункціональної точки зору система повинна бути зрозумілою, стабільною та придатною для щоденного використання. Інтерфейс має бути максимально простим, з мінімальною кількістю дій для досягнення основної мети — перегляду розкладу. Важливо, щоб система швидко реагувала на запити користувача, коректно обробляла помилки введення, не втрачала збережені налаштування та забезпечувала цілісність даних у базі. Конфігураційні параметри, зокрема токен Telegram-бота та службові налаштування, не повинні зберігатися у відкритому вигляді в коді та мають бути винесені до окремих конфігураційних файлів або змінних оточення. [12; 20]

Оскільки система орієнтована на навчальний проєкт і конкретну академічну групу, її архітектура може бути реалізована на основі відносно простої локальної бази даних і модульного підходу до програмної реалізації. Проте навіть у такому випадку потрібно передбачити можливість подальшого розвитку: розширення на інші групи, факультети, додавання нових типів занять, підтримку змін у розкладі, інтеграцію з навчальними ресурсами та більш розвинену систему нагадувань.

Таким чином, вимоги до програмної системи охоплюють не лише базовий перегляд розкладу, а й персоналізацію, автоматичну обробку навчальних даних, класифікацію занять, збереження користувацьких налаштувань, зручний інтерфейс і можливість подальшого масштабування. Це дозволяє розглядати систему не просто як довідковий інструмент, а як повноцінний програмний засіб підтримки навчальної діяльності студента.

1.3. Моделювання предметної області

Моделювання предметної області для програмного забезпечення автоматизації доступу до розкладу занять студентів дозволяє перейти від загального опису навчального процесу до узгодженої моделі функцій, ролей, даних та послідовностей дій, які підтримує система. Мета такого моделювання полягає у зменшенні неоднозначності вимог, визначенні меж відповідальності користувача та програмної системи, а також у підготовці основи для подальшого проєктування архітектури, бази даних, інтерфейсу користувача та програмних модулів. У роботі доцільно використовувати набір UML-діаграм і пов’язаних з ними графічних моделей, які взаємно доповнюють одна одну та дозволяють представити предметну область на різних рівнях деталізації. [16; 17]

З функціональної точки зору предметну область доцільно представити як множину прецедентів, тобто завершених користувацьких задач, які система повинна підтримувати. Для Telegram-бота розкладу такими прецедентами є запуск бота, перегляд розкладу, перегляд розкладу на сьогодні, завтра і тиждень, перегляд вибіркових дисциплін, зміна вибіркових дисциплін, перегляд профілю та налаштування нагадувань. Окремо виділяються системні прецеденти, що пов’язані з імпортом офіційного розкладу, визначенням типу навчального тижня, фільтрацією записів та формуванням персоналізованого результату. У такій моделі відношення <<include>> може використовуватись для уточнення, що загальний прецедент «Переглянути розклад» включає підпрецеденти «Переглянути розклад на сьогодні», «Переглянути розклад на завтра» та «Переглянути розклад на тиждень». Таким чином, діаграма прецедентів не лише перелічує функції системи, а й показує структуру доступу до основних можливостей бота. Основні прецеденти системи подано на рис. 1.

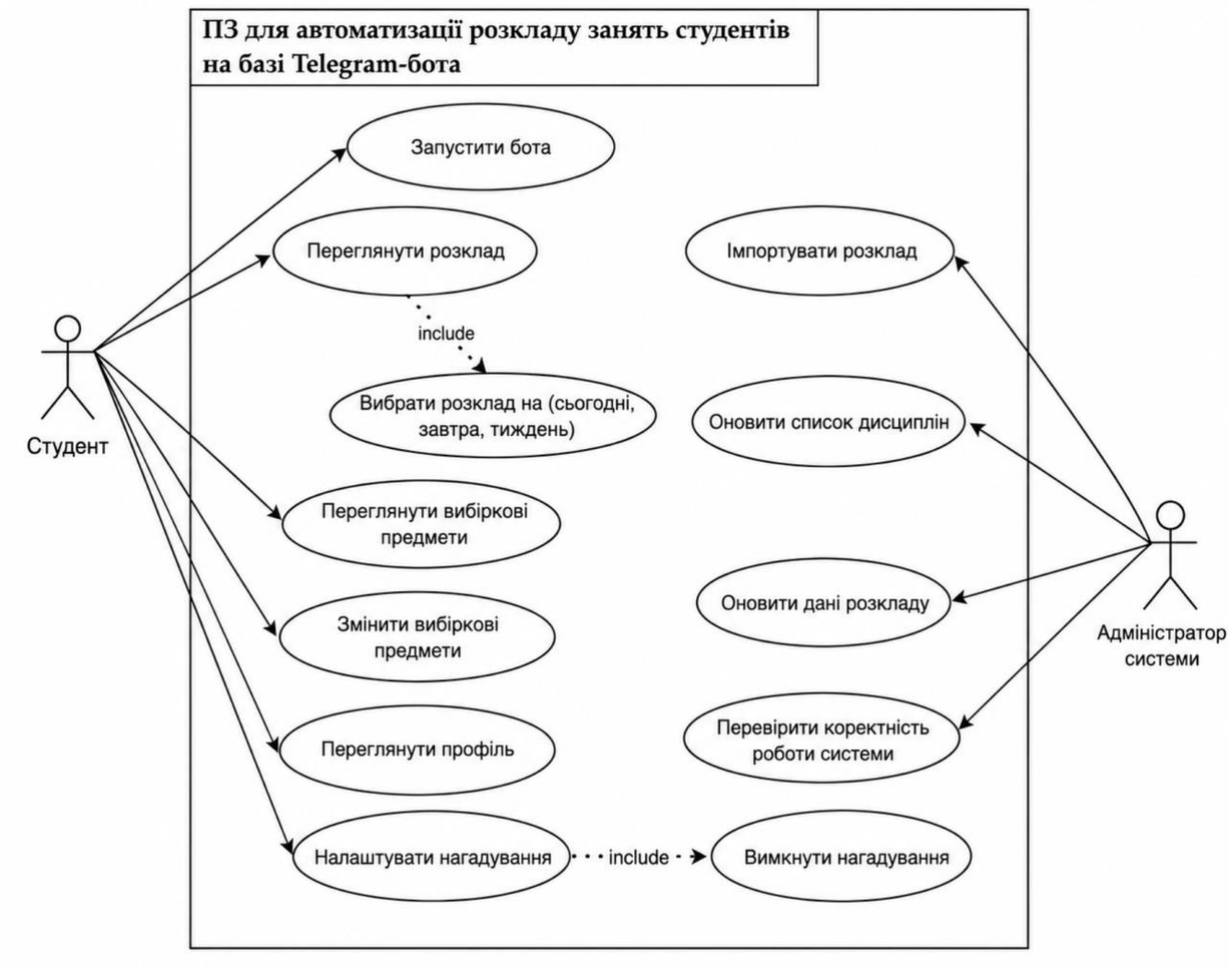

Рис. 1 Діаграма прецедентів

Після статичного опису функцій необхідно відобразити динаміку роботи системи, тобто те, як дії користувача та програмних модулів розгортаються в часі. Для цього використовується діаграма послідовності, на якій можуть бути представлені життєві лінії користувача, Telegram-бота, бази даних, модуля визначення типу тижня та модуля формування розкладу. Типовий сценарій включає вибір користувачем пункту «Розклад», отримання ботом даних про користувача, зчитування профілю та вибіркових дисциплін із бази даних, визначення поточного типу тижня, отримання записів розкладу та формування персоналізованої відповіді. Така діаграма дозволяє чітко показати, як саме відбувається взаємодія між інтерфейсом користувача, логікою застосунку та сховищем даних. Послідовність взаємодії користувача, Telegram-бота та бази даних наведено на рис. 2.

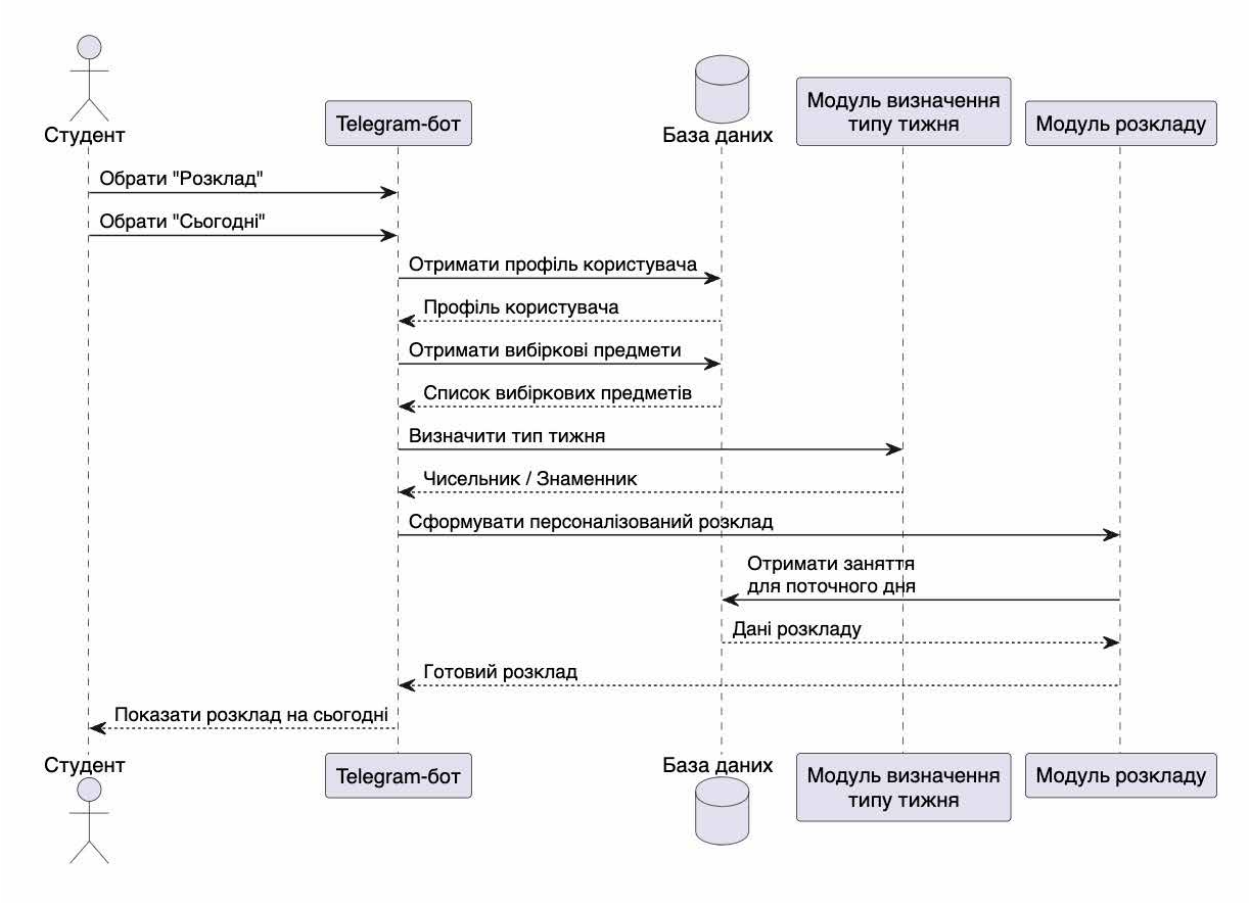

Рис. 2 Діаграма послідовностей

Окрім послідовності повідомлень, предметну область корисно описати як процес із умовами, перевірками та альтернативними сценаріями. Для цього використовується діаграма діяльності, яка відображає життєвий цикл роботи користувача з ботом. На такій діаграмі можуть бути представлені етапи запуску системи, вибору пункту меню, перевірки наявності профілю, визначення типу тижня, отримання даних розкладу, формування відповіді та відображення результату. Окремо в моделі можуть бути передбачені тупикові ситуації, наприклад відсутність профілю користувача, некоректне введення вибіркових дисциплін або відсутність занять на обраний день. Така форма подання дозволяє узгодити поведінку системи в нормальних та виняткових сценаріях і підкреслює процесний характер взаємодії користувача з ботом. Загальний процес роботи користувача з ботом відображено на рис. 3.

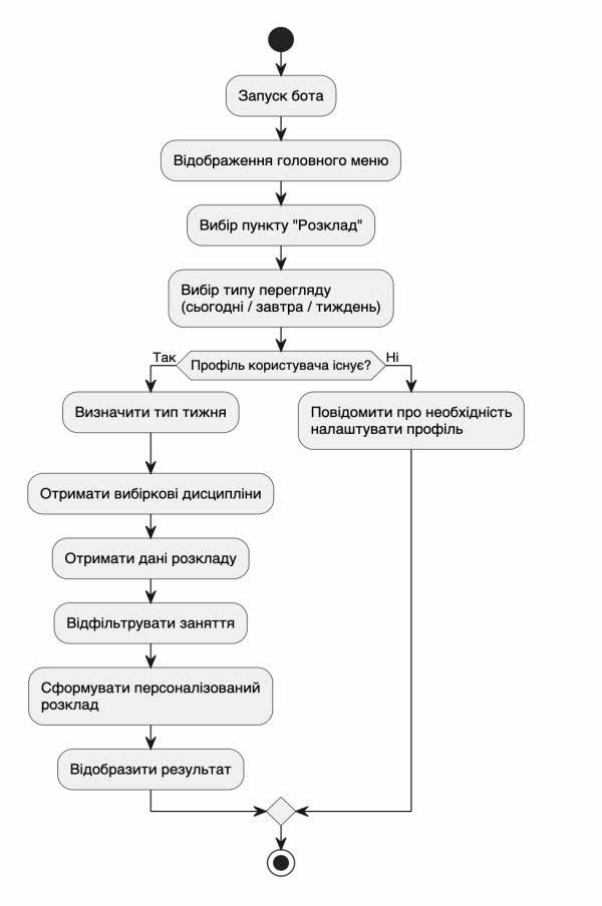

Рис. 3 Діаграма діяльності життєвого циклу взаємодії користувача з Telegram-ботом

Для узгодження основних понять предметної області доцільно використовувати діаграму абстракцій. На ній відображаються базові сутності системи, їх важливі властивості та обов’язки. У межах даної роботи такими абстракціями виступають користувач, Telegram-бот, розклад, навчальне заняття, вибіркова дисципліна та профіль користувача. Користувач є основним суб’єктом взаємодії із системою, Telegram-бот забезпечує доступ до функціональних можливостей, розклад об’єднує інформацію про заняття, а вибіркова дисципліна формує індивідуальну частину навчального навантаження студента. Така діаграма дозволяє визначити межі відповідальності окремих елементів системи та підготувати основу для переходу до проєктування класів, компонентів і структури бази даних. Основні абстракції предметної області подано на рис. 4.

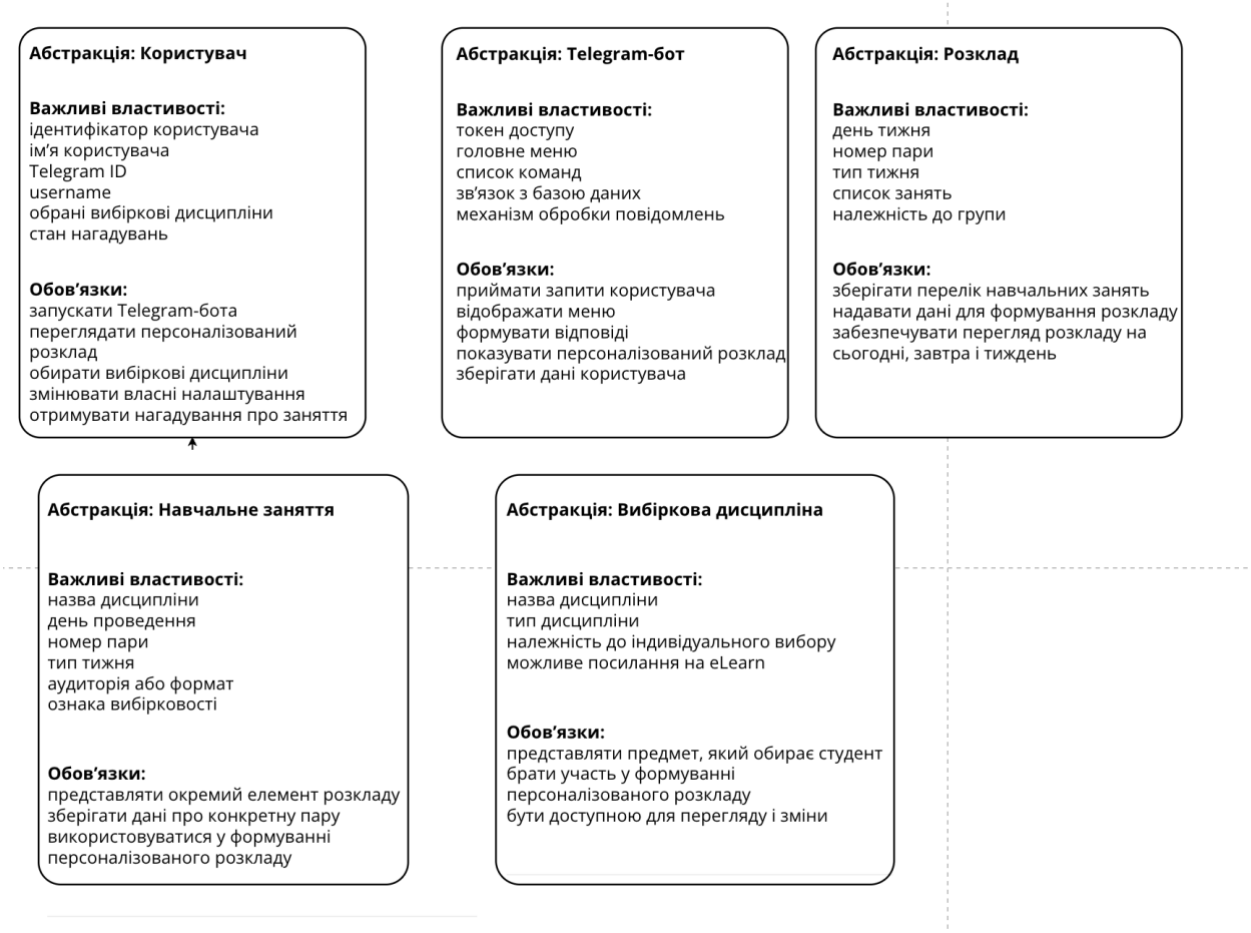

Рис. 4 Діаграма абстракцій предметної області

У сукупності чотири діаграми формують багаторівневе моделювання предметної області: прецеденти задають, що система робить, послідовність показує, у якому порядку відбувається взаємодія між користувачем і програмними модулями, діяльність описує, як процес розвивається з урахуванням умов, перевірок і завершення, а абстракції відображають, з яких основних понять складається предметна область. Це створює логічний місток до наступних розділів записки: аналізу існуючих рішень, постановки інженерного завдання та проєктування конкретної програмної реалізації, де окремі кроки діаграм відображаються в архітектурі Telegram-бота, структурі бази даних і програмних модулях системи.

1.4. Огляд існуючих рішень для автоматизації доступу до розкладу занять студентів

На сьогодні існує багато програмних рішень, що використовуються для організації доступу до розкладу занять у закладах вищої освіти. Такі рішення можуть бути реалізовані у вигляді офіційних вебсайтів університетів, електронних кабінетів студентів, мобільних застосунків, систем дистанційного навчання або ботів у месенджерах. Кожне з цих рішень має власну цільову аудиторію, набір функцій, переваги та обмеження. Їх аналіз дозволяє визначити, які можливості є типовими для сучасних систем доступу до розкладу, а також які недоліки доцільно врахувати під час розробки власного програмного забезпечення. [14]

Найпоширенішим варіантом доступу до розкладу є офіційні вебсайти університетів. Їх головною перевагою є централізоване зберігання та публікація навчальної інформації, доступність через браузер і відсутність потреби встановлювати додаткове програмне забезпечення. Як правило, такі сайти дозволяють переглядати розклад за факультетом, спеціальністю, курсом або групою. Водночас подібні рішення мають і недоліки: інтерфейс не завжди пристосований до мобільного використання, пошук потрібної групи може займати час, а сам розклад подається у загальному вигляді без персоналізації під конкретного студента. Крім того, у випадках наявності вибіркових дисциплін студент змушений самостійно визначати, які саме пари належать йому.

Іншим поширеним підходом є використання електронних кабінетів студентів або університетських інформаційних систем. Такі рішення можуть поєднувати розклад, оцінки, навчальні матеріали та персональні дані користувача. Їх перевагою є інтеграція кількох навчальних сервісів в одному середовищі. Проте подібні системи часто є складнішими в користуванні, мають перевантажений інтерфейс і орієнтовані на ширший спектр завдань, ніж простий швидкий доступ до розкладу. Для щоденного перегляду занять студенту іноді зручніше скористатися легшим інструментом.

Окрему групу становлять мобільні застосунки для навчальних закладів або універсальні студентські додатки. Їх сильною стороною є адаптація під смартфони, збереження даних користувача, можливість надсилання сповіщень і краща взаємодія з мобільною операційною системою. Разом із тим розробка і підтримка окремого мобільного застосунку потребує більших ресурсів, а користувачеві необхідно додатково встановлювати програму, реєструватися та стежити за її оновленням. Для невеликого навчального проєкту такий підхід є складнішим у реалізації порівняно з ботом у месенджері.

У сучасній практиці все більш популярними стають рішення на основі месенджерів, зокрема Telegram-боти. Їх перевагами є швидкий доступ, кросплатформеність, простота інтерфейсу та відсутність необхідності створювати окремий вебсайт чи мобільний застосунок для кінцевого користувача. Telegram-бот може надавати інформацію у форматі повідомлень, меню та кнопок, що є природним і зрозумілим для більшості студентів. Водночас такі рішення потребують чіткого проєктування діалогового інтерфейсу, продуманої структури меню та надійного зберігання даних, оскільки саме бот виступає основною точкою взаємодії користувача із системою. [3; 23]

Якщо розглядати рішення для доступу до розкладу саме з точки зору персоналізації, то більшість існуючих систем орієнтуються на груповий або потоковий розклад. Це означає, що користувач отримує повний перелік занять для академічної групи, а не власний персоналізований варіант. Такий підхід є достатнім у випадках, коли навчальний план однаковий для всіх студентів групи, але втрачає ефективність, коли з’являються вибіркові дисципліни. У цій ситуації особливу цінність має система, яка дозволяє зберігати індивідуальний вибір студента і на його основі формувати фактичний розклад.

Розроблюване у цій роботі програмне забезпечення не має на меті повністю замінити великі університетські інформаційні системи або багатофункціональні освітні платформи. Його перевага полягає в іншому: воно є простішим, зрозумілішим і краще пристосованим для щоденного використання конкретним студентом. На відміну від загальних вебсторінок розкладу, система

надає персоналізований розклад, враховує вибіркові дисципліни, автоматично визначає тип навчального тижня і може підтримувати нагадування про заняття. Завдяки використанню Telegram-бота користувач отримує доступ до інформації безпосередньо у звичному середовищі месенджера.

Перевагою власного програмного забезпечення є також можливість адаптації до конкретних умов навчального закладу та потреб окремої групи. У готових університетських системах користувач зазвичай працює з уже визначеним інтерфейсом і набором функцій. У власному рішенні можна змінювати логіку меню, спосіб відображення розкладу, структуру даних, механізм роботи з вибірковими дисциплінами, реалізацію нагадувань та додаткових посилань на навчальні ресурси. Це робить систему не лише зручним інструментом для користувача, а й наочним прикладом повного циклу розробки прикладного програмного забезпечення.

Таким чином, аналіз існуючих рішень показує, що наявні засоби доступу до розкладу переважно забезпечують або загальний перегляд розкладу, або інтеграцію в межах великих освітніх систем, але не завжди дають достатній рівень персоналізації та зручності. Саме тому доцільною є розробка Telegram-бота, який поєднує простоту використання, швидкий доступ, персоналізацію та можливість подальшого розширення функціональності.

1.5. Постановка завдання

На основі аналізу предметної області, вимог до програмної системи та огляду існуючих рішень було визначено завдання розробити програмне забезпечення для автоматизації доступу до персоналізованого розкладу занять студентів у вигляді Telegram-бота. Система має забезпечувати швидкий перегляд розкладу на сьогодні, завтра та тиждень, враховувати індивідуально обрані вибіркові дисципліни користувача, автоматично визначати тип навчального тижня та надавати зручний інтерфейс взаємодії через меню і повідомлення Telegram.

У межах роботи необхідно реалізувати серверну частину, яка відповідатиме за обробку команд користувача, формування відповідей, взаємодію з базою даних та імпорт офіційного розкладу з Excel-файлу. Клієнтська частина має бути реалізована у форматі Telegram-інтерфейсу та забезпечувати зручне головне меню, меню перегляду розкладу, роботу з вибірковими дисциплінами, профілем користувача та налаштуваннями нагадувань.

Окремим завданням є реалізація логіки визначення типу заняття. Система повинна вміти розрізняти лекційні та практичні заняття на основі структури вихідного Excel-файлу. Якщо запис про заняття є спільним для кількох академічних груп, його потрібно трактувати як лекційне заняття, а якщо подано окремо для кожної групи — як практичне. Для лекційних занять має бути передбачена можливість відображення додаткових посилань, зокрема на Google Meet.

Отже, основне завдання роботи полягає у створенні працездатного Telegram-бота, який поєднує персоналізований доступ до розкладу, роботу з вибірковими дисциплінами, підтримку нагадувань та інтеграцію з офіційним джерелом даних. Результатом виконання завдання має бути програмне забезпечення, придатне для демонстрації, тестування та подальшого розвитку.

1.6. Висновки до розділу

У першому розділі було розглянуто предметну область програмних засобів для автоматизації доступу до розкладу занять студентів. Визначено, що такі системи є актуальними для сучасного освітнього середовища, оскільки дозволяють студентам швидко отримувати актуальну інформацію про навчальні заняття, планувати власний час і зручніше взаємодіяти з навчальним процесом.

Було проаналізовано основні вимоги до програмної системи: можливість перегляду розкладу на сьогодні, завтра і тиждень, підтримка вибіркових дисциплін, автоматичне визначення типу навчального тижня, робота з профілем користувача та налаштування нагадувань. Окремо враховано нефункціональні вимоги, зокрема простоту інтерфейсу, зручність щоденного використання, коректну роботу з даними та можливість подальшого розширення системи.

Для кращого розуміння предметної області було використано моделювання: побудовані діаграми відображають основні функції системи, послідовність взаємодії користувача з Telegram-ботом, загальний процес роботи системи та базові абстракції предметної області. Це дозволило уточнити структуру майбутнього програмного забезпечення та підготувати основу для подальшого проєктування.

Також було виконано огляд існуючих рішень для доступу до розкладу занять студентів. Встановлено, що наявні засоби переважно орієнтовані на загальний перегляд розкладу або інтеграцію в межах великих інформаційних систем, але не завжди забезпечують достатній рівень персоналізації та зручності. Це підтверджує доцільність створення власного Telegram-бота, який поєднує простоту використання, швидкий доступ, персоналізацію розкладу та можливість подальшого розвитку.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Логічна модель даних у вигляді ER-діаграми

Логічна модель даних є важливою частиною проєктування інформаційного забезпечення, оскільки вона показує, які дані зберігаються в системі, як вони пов’язані між собою та які об’єкти предметної області необхідно врахувати під час подальшої реалізації бази даних. Для програмного забезпечення автоматизації доступу до розкладу занять студентів основними інформаційними об’єктами є користувачі, їхні профілі, вибіркові дисципліни, розклад, навчальні заняття, вибір користувача та нагадування.

У межах розроблюваної системи можна виділити кілька основних сутностей. Сутність «Користувачі» зберігає основні дані про студентів, які взаємодіють із Telegram-ботом. Вона є базовою для персоналізованих функцій системи, оскільки саме з користувачем пов’язані профіль, вибіркові дисципліни та нагадування. До цієї сутності належать такі атрибути, як Telegram ID, username, ім’я, прізвище та час реєстрації.

Сутність «Профілі» містить додаткову інформацію про користувача, зокрема стан нагадувань, поточний тип тижня та час оновлення профілю. Винесення цих даних в окрему сутність є доцільним, оскільки облікові дані користувача та його індивідуальні налаштування виконують різні ролі в системі. Між сутностями «Користувачі» та «Профілі» існує зв’язок типу «один до одного», оскільки кожному користувачу відповідає лише один профіль, а кожен профіль належить одному користувачеві.

Сутність «Вибіркові дисципліни» містить перелік доступних вибіркових предметів, які можуть бути обрані користувачем. Для кожної дисципліни зберігаються її назва, посилання на eLearn та статус активності. Оскільки один користувач може обрати кілька дисциплін, а одну дисципліну можуть обрати кілька користувачів, у системі використовується проміжна сутність «Вибір користувача», яка реалізує зв’язок типу «багато до багатьох» між користувачами

та вибірковими дисциплінами. У цій сутності зберігаються код користувача, код дисципліни та час вибору.

Сутність «Розклад» описує загальні параметри розкладу, зокрема назву групи, тип тижня, джерело даних і дату імпорту. Вона використовується як базова сутність для формування набору навчальних занять. Сутність «Навчальні заняття» містить окремі записи про пари: назву дисципліни, день тижня, номер пари, аудиторію або формат заняття та ознаку вибірковості. Один запис розкладу може містити багато навчальних занять, тому між сутностями «Розклад» та «Навчальні заняття» існує зв’язок типу «один до багатьох».

Сутність «Нагадування» описує повідомлення, які формуються системою для користувача. Вона містить код нагадування, код користувача, час надсилання, статус і текст повідомлення. Між сутностями «Користувачі» та «Нагадування» існує зв’язок типу «один до багатьох», оскільки один користувач може мати кілька нагадувань.

Зв’язки між сутностями відображають логіку роботи системи. Користувач пов’язаний із профілем відношенням «один до одного». Користувач пов’язаний із нагадуваннями відношенням «один до багатьох». Між користувачами та вибірковими дисциплінами існує зв’язок «багато до багатьох», який реалізується через сутність «Вибір користувача». Сутність «Розклад» пов’язана із сутністю «Навчальні заняття» відношенням «один до багатьох», оскільки кожен розклад складається з набору окремих пар. Така структура дозволяє зберігати як загальні дані розкладу, так і персоналізовані параметри конкретного студента. ER-діаграму логічної моделі даних наведено на рис. 5.

Рис. 5 ER-діаграма логічної моделі даних

Логічна модель даних, зображена на ER-діаграмі, є основою для подальшого створення фізичної структури бази даних. На її основі можна визначити таблиці, первинні та зовнішні ключі, типи зв’язків і обмеження цілісності. Запропонована модель є достатньою для реалізації основних функцій системи: збереження користувачів, профілів, вибіркових дисциплін, записів розкладу та нагадувань. Надалі ця модель може бути розширена додатковими сутностями, наприклад посиланнями на Google Meet, журналом імпорту розкладу або історією змін у розкладі.

2.2. Вибір системи управління базами даних

Для розроблюваної системи автоматизації доступу до розкладу занять студентів важливо обрати таку систему управління базами даних, яка забезпечить надійне збереження інформації про користувачів, їхні профілі, вибіркові дисципліни, записи розкладу та нагадування. Оскільки програмне забезпечення реалізується у вигляді Telegram-бота і має навчальний характер, база даних повинна бути простою в налаштуванні, зручною для інтеграції з Python та достатньою для роботи зі зв’язаними сутностями. [14]

У межах роботи доцільно розглянути кілька поширених варіантів, зокрема MySQL, MongoDB, PostgreSQL та SQLite. MySQL є популярною реляційною системою управління базами даних, має високу швидкодію, широку сферу використання та добре підходить для класичних вебзастосунків. Проте для навчального проєкту її використання потребує окремого серверного налаштування, що ускладнює розгортання і супровід системи.

MongoDB є документоорієнтованою базою даних і добре підходить для зберігання гнучких JSON-подібних структур. Її доцільно використовувати там, де схема даних часто змінюється або де зв’язки між об’єктами є менш формалізованими. Однак у розроблюваній системі основні сутності мають чіткі зв’язки: користувачі, профілі, вибіркові дисципліни, вибір користувача, розклад, навчальні заняття та нагадування. Тому реляційна модель у цьому випадку є більш природною та зрозумілою.

PostgreSQL є потужною реляційною СУБД з відкритим кодом, яка підтримує зовнішні ключі, обмеження цілісності, транзакції та розширені можливості роботи з даними. Вона добре підходить для систем, де важливо зберігати зв’язану інформацію. Разом із тим для невеликого локального навчального проєкту повноцінне розгортання PostgreSQL є більш складним порівняно з легшими рішеннями. [25]

У цій роботі обрано SQLite, оскільки вона є вбудованою реляційною системою управління базами даних, не потребує окремого серверного встановлення, легко інтегрується з мовою програмування Python та забезпечує

достатню функціональність для реалізації навчального програмного продукту. SQLite дозволяє зберігати базу даних у вигляді одного файлу, що спрощує розробку, тестування, резервне копіювання та перенесення системи між середовищами. [4]

Важливою перевагою SQLite є підтримка реляційної моделі даних, що дозволяє коректно реалізувати таблиці користувачів, профілів, вибіркових дисциплін, записів розкладу та нагадувань, а також зв’язки між ними. Для розроблюваного Telegram-бота цього достатньо, оскільки система орієнтована на відносно невелику кількість користувачів і не потребує високонавантаженої серверної інфраструктури. Крім того, використання SQLite скорочує час налаштування середовища та дозволяє зосередитися на основній логіці роботи системи. [4]

Для реалізації інформаційного забезпечення було обрано SQLite. Такий вибір є обґрунтованим, оскільки поєднує простоту використання, надійність реляційної моделі, зручність інтеграції з Python і достатню функціональність для навчального проєкту. Це відповідає потребам розроблюваної системи та спрощує її подальше тестування, демонстрацію і супровід. [4]

2.3. Розробка структури інформаційної бази

Структура інформаційної бази розроблюваної системи формується на основі логічної моделі даних, описаної в попередніх підрозділах. Її основне призначення — забезпечити збереження даних, необхідних для роботи користувачів із Telegram-ботом: облікової інформації, профілів, вибіркових дисциплін, записів розкладу та нагадувань. Оскільки система орієнтована на навчальний Telegram-сервіс і використовує локальну базу даних SQLite, структура інформаційної бази проєктується у вигляді реляційних таблиць.

Основою інформаційної бази є таблиця користувачів, у якій зберігаються базові відомості про студентів, що взаємодіють із ботом. До таких даних належать унікальний код користувача, Telegram ID, username, ім’я, прізвище та час реєстрації. Саме ця таблиця є центральною для інших сутностей, оскільки з нею пов’язані профілі, вибіркові дисципліни та нагадування.

Для збереження індивідуальних налаштувань користувача використовується таблиця профілів. Вона містить стан нагадувань, поточний тип тижня та час оновлення профілю. Такий підхід дозволяє відокремити базові облікові дані від персональних параметрів, які використовуються під час формування індивідуального розкладу. Таблиця профілів пов’язується з таблицею користувачів за кодом користувача, що забезпечує відповідність одного профілю одному користувачеві.

Окремою частиною інформаційної бази є таблиця вибіркових дисциплін, яка містить перелік доступних вибіркових предметів. Для кожної дисципліни зберігаються її назва, посилання на eLearn та статус активності. Оскільки один користувач може обирати кілька дисциплін, а одну й ту саму дисципліну можуть обрати кілька користувачів, у структурі бази передбачена проміжна таблиця вибір користувача. Вона реалізує зв’язок між користувачем і дисципліною та зберігає час вибору. Таким чином забезпечується можливість формування персоналізованого розкладу відповідно до індивідуального навчального плану студента.

Для збереження загальних параметрів розкладу використовується таблиця розклад, у якій фіксуються назва групи, тип тижня, джерело даних і дата імпорту. Вона є базовою сутністю для таблиці навчальні заняття, у якій зберігаються окремі записи про пари: назва дисципліни, день тижня, номер пари, аудиторія або формат проведення, а також ознака вибірковості. Такий поділ є доцільним, оскільки дозволяє відокремити загальну інформацію про розклад від конкретних елементів навчального процесу.

Для реалізації функції нагадувань використовується таблиця нагадування, яка містить код нагадування, код користувача, час надсилання, статус і текст повідомлення. Ці дані дозволяють формувати та зберігати повідомлення, що надсилаються студенту у визначений момент часу. Оскільки один користувач може мати кілька нагадувань, таблиця нагадувань пов’язана з таблицею користувачів через зовнішній ключ.

Зв’язки між таблицями будуються за принципами реляційної моделі. Таблиця профілів має зв’язок один до одного з таблицею користувачів. Таблиця нагадувань пов’язана з користувачами зв’язком один до багатьох. Між таблицями користувачів і вибіркових дисциплін реалізовано зв’язок багато до багатьох, який забезпечується через таблицю вибору користувача. Таблиця розкладу пов’язана з таблицею навчальних занять зв’язком один до багатьох, оскільки один розклад включає багато окремих пар.

Під час розробки структури інформаційної бази важливо враховувати цілісність даних. Для цього використовуються первинні ключі, що однозначно ідентифікують записи, та зовнішні ключі, які забезпечують коректність зв’язків між таблицями. Наприклад, нагадування не повинно посилатися на неіснуючого користувача, а запис про вибір дисципліни має відповідати як реальному користувачу, так і реальній дисципліні. Також доцільно використовувати обмеження унікальності, щоб уникнути дублювання вибору однієї й тієї ж дисципліни одним користувачем.

Отже, розроблена структура інформаційної бази забезпечує збереження основних даних, необхідних для роботи системи: користувачів, профілів,

вибіркових дисциплін, записів розкладу та нагадувань. Така структура є достатньою для реалізації базових функцій Telegram-бота й водночас може бути розширена в майбутньому, наприклад шляхом додавання посилань на Google Meet, журналу імпорту розкладу, історії змін у розкладі або додаткових навчальних ресурсів.

2.4. Схема та фізична структура бази даних

Фізична структура бази даних є практичним продовженням логічної моделі, описаної в попередніх підрозділах. Якщо логічна модель показує основні сутності та зв’язки між ними, то фізична структура визначає, які саме таблиці створюються в базі даних, які поля вони містять, які типи даних використовуються та яким чином забезпечується цілісність інформації.

У межах розроблюваної системи база даних реалізується на основі SQLite. Такий підхід дозволяє поєднати простоту розгортання, реляційну структуру даних та зручну інтеграцію з мовою програмування Python. Основними таблицями фізичної структури є users, user_profiles, elective_subjects, user_elective_subjects, schedule_items та reminders.

Таблиця users відповідає за збереження основних даних користувачів Telegram-бота. Вона містить унікальний ідентифікатор користувача, telegram_id, username, first_name, last_name і created_at. Саме ця таблиця є центральною в структурі бази даних, оскільки з нею пов’язані профілі, вибіркові дисципліни та нагадування.

Таблиця user_profiles використовується для збереження додаткової інформації про користувача. До неї входять group_name, reminders_enabled, current_week_type та updated_at. Поле user_id у цій таблиці одночасно є первинним і зовнішнім ключем, що забезпечує зв’язок типу один до одного між таблицями users та user_profiles. Такий підхід є доцільним, оскільки технічна ідентифікація користувача та його індивідуальні параметри виконують різні ролі в системі

Таблиця elective_subjects призначена для збереження переліку вибіркових дисциплін. Вона містить ідентифікатор дисципліни, її назву, посилання на eLearn та ознаку активності. Для реалізації зв’язку між користувачами та вибірковими дисциплінами використовується таблиця user_elective_subjects, яка містить поля id, user_id, subject_id та selected_at. Саме ця таблиця реалізує зв’язок типу багато до багатьох між користувачами та вибірковими дисциплінами. Для запобігання

дублюванню вибору однієї й тієї ж дисципліни одним користувачем доцільно використовувати обмеження унікальності для пари user_id і subject_id.

Таблиця schedule_items використовується для збереження імпортованих записів офіційного розкладу. У ній містяться такі поля, як group_name, day_of_week,pair_number,week_type,raw_subject_name,normalized_subject_name, room,teacher,is_elective та imported_at. Ця таблиця є основою для формування розкладу на сьогодні, завтра або тиждень. На поточному етапі вона не має зовнішніх ключів до інших таблиць, оскільки використовується як окреме джерело даних для подальшої фільтрації та формування персоналізованого розкладу на рівні програмної логіки.

Таблиця reminders призначена для реалізації нагадувань. Вона містить ідентифікатор нагадування, посилання на користувача, час надсилання, статус та текст повідомлення. Така структура дозволяє зберігати параметри нагадувань і забезпечує можливість надсилання індивідуальних повідомлень конкретному користувачеві. Схему фізичної структури бази даних системи наведено на рис. 6.

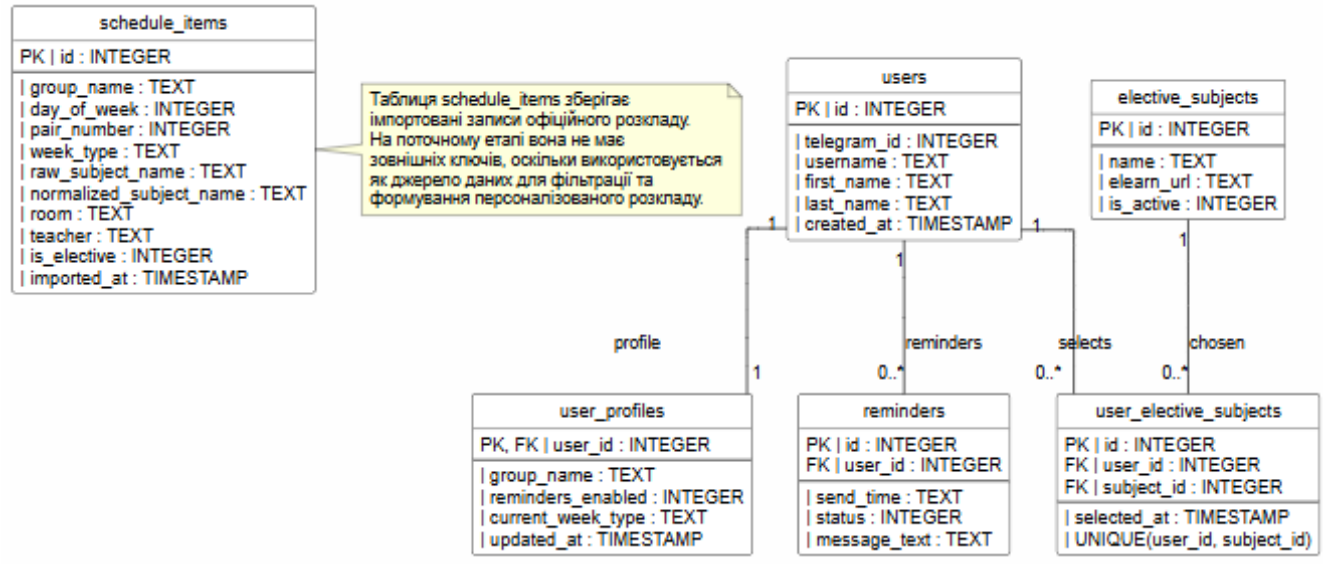

Рис. 6 Схема фізичної структури бази даних системи

Зв’язки між таблицями реалізуються за допомогою первинних і зовнішніх ключів. Первинні ключі однозначно ідентифікують кожний запис у таблиці, а зовнішні ключі забезпечують правильний зв’язок між сутностями. Наприклад, запис у таблиці user_profiles має відповідати конкретному користувачу з таблиці

users, таблиця reminders пов’язується з користувачем через поле user_id, а таблиця user_elective_subjects одночасно посилається і на користувача, і на вибіркову дисципліну.

Для забезпечення коректності даних доцільно використовувати обмеження цілісності. Наприклад, кожний користувач повинен мати лише один профіль, а один і той самий користувач не повинен мати дубльований запис про вибір однієї й тієї ж дисципліни. Також важливо забезпечити коректність значень типу тижня, статусу нагадування та інших службових полів.

Фізична структура бази даних повинна враховувати і практичні потреби системи. Оскільки розроблюваний програмний продукт реалізується як Telegram-бот, база даних має забезпечувати швидкий доступ до профілю користувача, збереження його вибіркових дисциплін, формування персоналізованого розкладу та підтримку нагадувань. Запропонована структура є достатньою для реалізації базових функцій системи й може бути розширена в майбутньому, наприклад шляхом додавання таблиць для посилань на Google Meet, журналу імпорту розкладу або історії змін у розкладі.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РОЗКЛАДУ ЗАНЯТЬ СТУДЕНТІВ

3.1. Моделювання структури та взаємодії об'єктів

Після визначення вимог і проєктування інформаційної бази наступним етапом є моделювання структури програмного забезпечення. На цьому етапі необхідно показати, з яких основних частин складається система, які об’єкти беруть участь у роботі Telegram-бота та як вони взаємодіють між собою під час формування персоналізованого розкладу. Для цього доцільно використати кілька типів діаграм: діаграму пакетів, діаграму класів та діаграму розгортання. Разом вони дозволяють описати систему з різних боків: логічну структуру коду, основні сутності та зв’язки між ними, а також фізичне розміщення компонентів у середовищі виконання.

Розроблюване програмне забезпечення має архітектуру, орієнтовану на роботу через Telegram. Основними частинами системи є клієнтський рівень Telegram, сервер Telegram-бота та рівень даних і зовнішніх джерел. Клієнтський рівень відповідає за взаємодію користувача з ботом через інтерфейс Telegram і Telegram API. Серверна частина виконує роль центрального координатора: вона обробляє команди користувача, реалізує бізнес-логіку, формує відповіді, імпортує розклад і працює з базою даних. Рівень даних і зовнішніх джерел забезпечує збереження інформації в SQLite та доступ до офіційного файлу розкладу і джерела розкладу НУБіП.

Першою доцільно розглянути діаграму пакетів, оскільки вона показує загальну організацію системи та розподіл її частин за відповідальністю. На клієнтському рівні виділяються інтерфейс користувача та Telegram API. Інтерфейс користувача представляє повідомлення, меню та кнопки, з якими працює студент. Telegram API забезпечує передачу запитів від користувача до серверної частини бота. На серверному рівні розташовуються bot.py як точка входу, модулі handlers, logic/services, keyboards, config, scheduler, parser та database module. Модуль handlers відповідає за обробку команд і повідомлень,

logic/services містить бізнес-логіку системи, keyboards формує меню й кнопки, parser виконує імпорт розкладу, scheduler відповідає за нагадування, а database module реалізує доступ до бази даних. Окремо виділено рівень даних та зовнішніх джерел, до якого входять SQLite DB, Excel-файл розкладу та джерело розкладу НУБіП. Діаграму пакетів програмного забезпечення наведено на рис. 7.

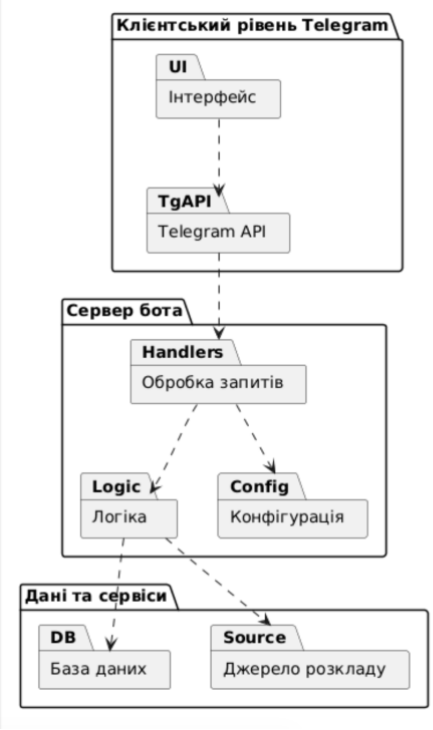

Рис. 7 Діаграма пакетів програмного забезпечення

Діаграма пакетів демонструє, що система побудована за принципом поділу відповідальності між трьома основними рівнями: клієнтським рівнем Telegram, сервером бота та рівнем даних і сервісів. Клієнтський рівень включає інтерфейс користувача та Telegram API, через які здійснюється взаємодія студента з системою. Сервер бота містить основні програмні пакети: Handlers, Logic і Config. Пакет Handlers відповідає за обробку запитів користувача, пакет Logic реалізує основну логіку роботи системи, а пакет Config зберігає службові налаштування програми. Рівень даних і сервісів представлений пакетами DB і Source, де DB відповідає за роботу з базою даних, а Source — за джерело розкладу. Такий поділ є зручним для супроводу, оскільки зміни в інтерфейсі, логіці чи структурі даних можна виконувати відносно незалежно.

Наступним елементом моделювання є діаграма класів. Вона дозволяє описати основні об’єкти предметної області та взаємозв’язки між ними. У межах системи ключовими сутностями є Користувач, ПрофільКористувача, TelegramБот, Розклад, НавчальнеЗаняття, ВибірковаДисципліна, Нагадування, ТипНавчальногоТижня та ОфіційнийФайлРозкладу. Користувач є центральною сутністю системи, оскільки саме він запускає бота, переглядає розклад, змінює вибіркові дисципліни та переглядає профіль. ПрофільКористувача зберігає налаштування користувача, Нагадування відповідає за повідомлення, ВибірковаДисципліна описує індивідуальний вибір студента, а Розклад та НавчальнеЗаняття відображають навчальний процес. Окремо виділено клас ТипНавчальногоТижня, який відповідає за визначення чисельника або знаменника, та клас ОфіційнийФайлРозкладу, що відображає джерело імпорту даних. Діаграму класів системи автоматизації доступу до розкладу занять наведено на рис. 8.

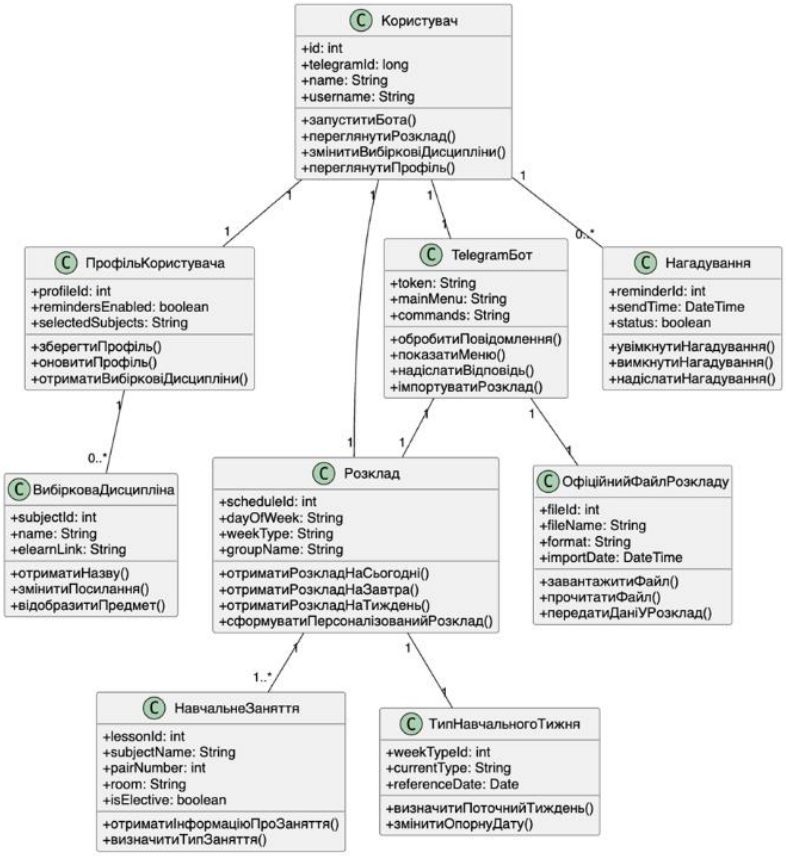

Рис. 8 Діаграма класів системи автоматизації доступу до розкладу занять

Діаграма класів допомагає краще зрозуміти, які об’єкти повинні бути представлені у програмній логіці. Наприклад, один користувач має один профіль, може мати кілька нагадувань і взаємодіє з Telegram-ботом для отримання інформації. Розклад містить набір навчальних занять, а вибіркові дисципліни впливають на формування персоналізованого розкладу. Така модель дозволяє логічно розділити відповідальність між класами: користувач відповідає за дії в системі, Telegram-бот — за координацію взаємодії, розклад — за структуру занять, а тип навчального тижня — за коректний вибір актуальних записів.

Окремо важливо описати взаємодію об’єктів під час роботи системи. Типовий сценарій починається з того, що користувач відкриває Telegram і надсилає команду або натискає кнопку меню. Далі через Telegram API запит потрапляє до сервера Telegram-бота, де bot.py передає його у відповідний модуль handlers. Після цього система звертається до модулів бізнес-логіки, бази даних та, за потреби, до модуля імпорту розкладу. У результаті формується відповідь, яка повертається користувачу через Telegram API. Якщо мова йде про нагадування, додатково задіюється модуль scheduler, а якщо про оновлення розкладу — модуль parser.

Третім важливим елементом є діаграма розгортання, яка показує, де фізично або логічно розміщуються компоненти системи під час виконання. На ній відображено смартфон студента з встановленим Telegram-клієнтом, хмару Telegram, сервер Telegram-бота, середовище виконання Python 3 Runtime, базу даних SQLite та зовнішнє джерело даних. Сервер Telegram-бота є центральним вузлом системи: він взаємодіє з Telegram Bot API, звертається до SQLite DB для збереження і читання даних, а також отримує розклад із зовнішнього джерела через сайт НУБіП або Excel-файл розкладу. У середовищі Python виконуються основні програмні артефакти: bot.py, handlers, services, scheduler.py та database.py. Діаграму розгортання програмної системи автоматизації доступу до розкладу занять наведено на рис. 9.

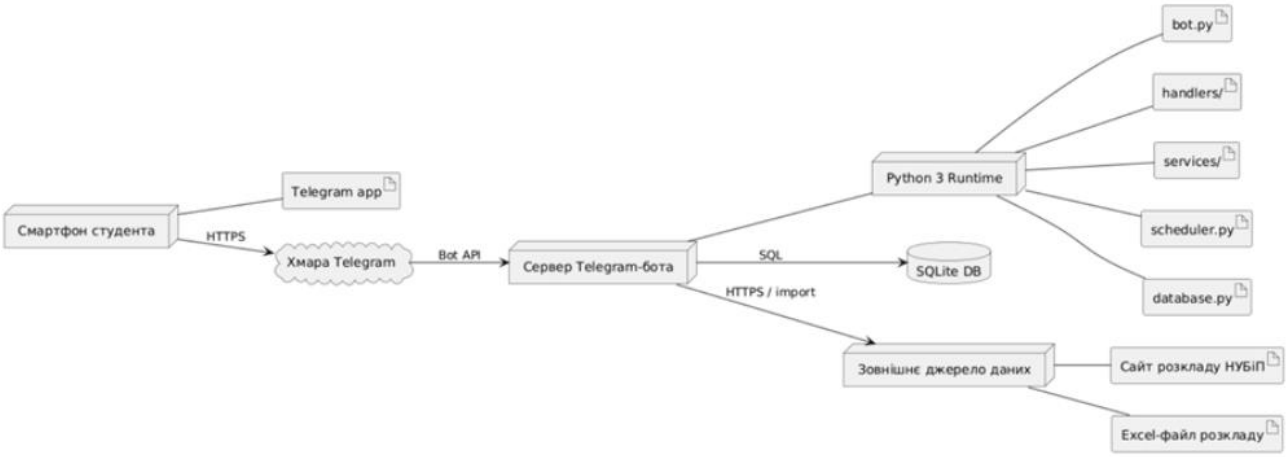

Рис. 9 Діаграма розгортання програмної системи автоматизації доступу до розкладу занять

Діаграма розгортання підкреслює важливу особливість обраної архітектури: користувач працює із системою через звичне середовище Telegram, а вся основна логіка зосереджена на сервері бота. Сервер не лише формує відповіді, а й виконує імпорт даних, роботу з базою та надсилання нагадувань. Такий підхід є обґрунтованим для навчального проєкту, оскільки дозволяє побудувати працездатну систему без створення окремого вебклієнта чи мобільного застосунку, зберігаючи при цьому модульність і можливість подальшого розвитку.

Загалом моделювання структури та взаємодії об’єктів показує, що система побудована за зрозумілим принципом поділу відповідальності. Telegram-клієнт відповідає за інтерфейс користувача, сервер Telegram-бота — за обробку запитів, бізнес-логіку, імпорт розкладу та нагадування, а SQLite і зовнішні джерела — за збереження й отримання даних. Таке розділення спрощує розробку, тестування та подальше розширення системи. Наприклад, у майбутньому можна додати журнал змін розкладу, посилання на Google Meet, інтеграцію з навчальними ресурсами або підтримку кількох академічних груп без повної перебудови архітектури системи.

3.2. Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення визначає, як саме розподілені основні частини системи, за що відповідає кожна з них і яким чином вони взаємодіють між собою. Для Telegram-бота автоматизації доступу до розкладу занять така структура є особливо важливою, оскільки система одночасно працює з інтерфейсом користувача, обробкою команд, бізнес-логікою, імпортом розкладу, нагадуваннями та базою даних.

Розроблюване програмне забезпечення побудоване за клієнт-серверним принципом. Клієнтська частина реалізується у середовищі Telegram і відповідає за взаємодію користувача з ботом через повідомлення, кнопки та меню. Серверна частина забезпечує обробку команд, формування відповідей, реалізацію бізнес-логіки, імпорт офіційного розкладу, роботу з нагадуваннями та збереження даних. Окремим рівнем виступають зовнішні джерела даних, зокрема офіційний Excel-файл розкладу та сайт розкладу НУБіП, а також локальна база даних SQLite.

Клієнтська частина програмного забезпечення включає інтерфейс користувача Telegram і Telegram API. Інтерфейс користувача реалізує головне меню, меню перегляду розкладу, роботу з вибірковими дисциплінами, профілем та нагадуваннями. Telegram API забезпечує передавання повідомлень між користувачем і серверною частиною системи. Завдяки цьому користувач може працювати з ботом без встановлення окремого застосунку або відкриття вебсайту.

Серверна частина виконує роль центрального координатора. Головний файл bot.py є точкою входу до програми та ініціалізує основні модулі системи. Модулі handlers відповідають за обробку команд і повідомлень, logic/services реалізують бізнес-логіку, keyboards формують меню й кнопки, scheduler забезпечує роботу нагадувань, parser виконує імпорт розкладу з офіційного джерела, а database module відповідає за взаємодію з базою даних. Окремо виділено модуль config, який містить службові параметри системи.

База даних SQLite використовується для збереження користувачів, профілів, вибіркових дисциплін, записів розкладу та нагадувань. Такий підхід дозволяє централізовано працювати з даними, не дублювати інформацію та забезпечувати персоналізацію відповідей бота. Імпортовані з Excel-файлу записи розкладу зберігаються в структурованому вигляді й надалі використовуються під час формування персоналізованого розкладу для конкретного студента.

Основний сценарій взаємодії між частинами системи виглядає так: користувач надсилає команду або натискає кнопку в Telegram, Telegram API передає запит серверу бота, після чого відповідний модуль handlers обробляє його та звертається до бізнес-логіки. Далі система отримує необхідні дані з бази або, якщо потрібно, із зовнішнього джерела розкладу, формує відповідь і надсилає її користувачу. Якщо мова йде про нагадування, додатково активується модуль scheduler, який у визначений момент часу самостійно формує й відправляє повідомлення.

Особливістю організаційної структури є те, що система не потребує окремого вебклієнта чи мобільного застосунку. Уся взаємодія з користувачем відбувається через Telegram, що спрощує доступ до функцій системи та зменшує складність розробки інтерфейсу. При цьому основна логіка зосереджена на сервері бота, що дозволяє зручно допрацьовувати функціональність, не змінюючи повністю всю систему.

Організаційна структура програмного забезпечення складається з трьох основних рівнів: клієнтського, серверного та рівня даних і зовнішніх джерел. Клієнтський рівень забезпечує взаємодію з користувачем у середовищі Telegram, серверний — обробку запитів, бізнес-логіку, імпорт розкладу та нагадування, а рівень даних і зовнішніх джерел — збереження інформації та доступ до офіційного розкладу. Такий поділ робить систему зрозумілою, достатньо гнучкою та придатною для подальшого розвитку.

3.3. Компонентна структура системи

Компонентна структура системи показує, з яких великих функціональних частин складається програмне забезпечення та як ці частини взаємодіють між собою. На відміну від опису окремих класів або файлів, компонентна модель розглядає систему на вищому рівні — як набір взаємопов’язаних блоків, кожен з яких виконує власну роль. Для Telegram-бота автоматизації доступу до розкладу занять це особливо важливо, оскільки система одночасно включає інтерфейс користувача, обробку команд, бізнес-логіку, імпорт офіційного розкладу, нагадування та роботу з базою даних.

У розроблюваній системі можна виділити такі основні компоненти: клієнтський рівень Telegram, сервер Telegram-бота, модуль роботи з базою даних, модуль імпорту розкладу, модуль нагадувань та зовнішні джерела даних. Загальна компонентна структура наведена на діаграмі. Компонентну структуру системи наведено на рис. 10.

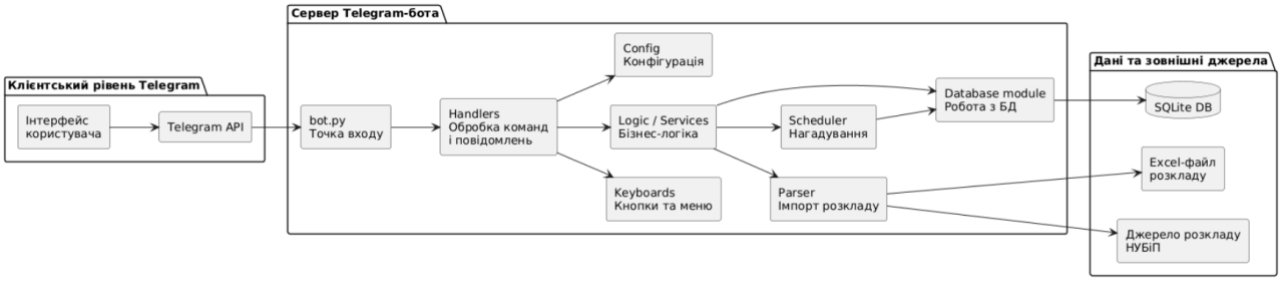

Рис. 10 Діаграма компонентів програмного забезпечення

Компонент «Інтерфейс користувача» працює на стороні Telegram і є основною частиною, з якою безпосередньо взаємодіє студент. Він відповідає за відображення вітальних повідомлень, кнопок меню, розділів перегляду розкладу, вибіркових дисциплін, профілю та нагадувань. Через Telegram API повідомлення користувача передаються до серверної частини, а сформовані відповіді повертаються назад у чат.

Компонент bot.py виконує роль точки входу до програми. Саме він ініціалізує основні модулі системи, забезпечує запуск бота та зв’язок між окремими частинами програмного забезпечення. Через нього відбувається

підключення конфігурації, модулів обробки команд, логіки роботи з розкладом, бази даних та нагадувань.

Компонент handlers відповідає за обробку команд і повідомлень користувача. Саме тут реалізується логіка переходу між пунктами меню, запуск окремих сценаріїв, обробка натискань на кнопки та передавання керування до відповідних модулів бізнес-логіки. Handlers є проміжною ланкою між інтерфейсом користувача та внутрішніми сервісами системи.

Компонент logic / services виконує функції бізнес-логіки. Він відповідає за формування персоналізованого розкладу, визначення поточного типу навчального тижня, застосування правил для вибіркових дисциплін, підготовку текстів повідомлень та загальну координацію внутрішньої логіки системи. Саме в цьому компоненті зосереджено основні алгоритми, які забезпечують практичну корисність Telegram-бота.

Компонент keyboards забезпечує формування кнопок і меню, які бачить користувач у Telegram. Він відповідає за побудову головного меню, підменю розкладу, меню роботи з вибірковими дисциплінами та інших елементів навігації. Виділення цього компонента в окремий модуль дозволяє змінювати інтерфейс без втручання в бізнес-логіку.

Компонент scheduler призначений для реалізації нагадувань. Він виконує перевірку часу, отримує відповідні дані користувача з бази та формує повідомлення про заняття. Завдяки цьому система може не лише реагувати на запити користувача, а й ініціювати взаємодію самостійно, надсилаючи повідомлення у визначений момент часу.

Компонент parser відповідає за імпорт офіційного розкладу. Він працює з Excel-файлом розкладу та джерелом розкладу НУБіП, виконує зчитування записів, нормалізацію назв дисциплін, визначення типу заняття та передавання отриманих даних до модуля роботи з базою даних. Цей компонент є критично важливим, оскільки саме він забезпечує актуальність інформації, яку надалі використовує Telegram-бот.

Компонент database module забезпечує взаємодію з базою даних SQLite. Через нього реалізується запис і читання користувачів, профілів, вибіркових дисциплін, записів розкладу та нагадувань. Саме цей компонент виконує роль посередника між бізнес-логікою та фізичним сховищем даних.

Окремо виділяється компонент config, який містить службові параметри системи, зокрема налаштування токена бота, параметри середовища, опорну дату для визначення типу тижня та інші значення, які не повинні бути жорстко вбудовані в основну логіку програми.

До рівня даних і зовнішніх джерел належать SQLite DB, Excel-файл розкладу та джерело розкладу НУБіП. SQLite DB забезпечує локальне збереження інформації, Excel-файл виступає джерелом офіційних записів розкладу, а сайт розкладу НУБіП є первинним зовнішнім джерелом навчальних даних. Таким чином, система спирається одночасно на внутрішнє сховище та зовнішнє джерело оновлення інформації.

Між компонентами існують чітко визначені зв’язки. Користувач взаємодіє з Telegram API, через яке запити надходять до bot.py. Далі керування передається в handlers, а звідти — до модулів бізнес-логіки, клавіатур, імпорту розкладу, нагадувань і бази даних. Компонент logic / services координує звернення до scheduler, parser та database module, а parser і database module взаємодіють із зовнішніми джерелами та базою даних. Така організація забезпечує чіткий поділ відповідальності та спрощує супровід системи.

Перевагою такої компонентної структури є те, що кожен функціональний блок можна розробляти, тестувати й удосконалювати окремо. Наприклад, можна змінити структуру меню без втручання у модуль імпорту розкладу, удосконалити нагадування без зміни обробки команд або розширити базу даних без зміни інтерфейсної частини. Крім того, така структура є зручною для подальшого розвитку системи: у майбутньому можна додати модуль аналізу змін у розкладі, підтримку кількох груп, посилання на Google Meet, інтеграцію з eLearn або інші освітні функції.

Компонентна структура розроблюваного програмного забезпечення відображає логічне розділення системи на основні функціональні блоки. Вона поєднує клієнтський рівень Telegram, сервер Telegram-бота, модулі бізнес-логіки, роботи з базою даних, імпорту розкладу та зовнішні джерела навчальної інформації. Такий підхід забезпечує достатню гнучкість, простоту супроводу та можливість подальшого розширення системи відповідно до майбутніх потреб.

3.4. Вибір інструментарію для створення прикладного програмного забезпечення

Вибір інструментарію для створення прикладного програмного забезпечення є важливим етапом розробки, оскільки саме від обраних технологій залежить зручність реалізації, стабільність роботи системи, можливість подальшого розширення та простота супроводу. Для Telegram-бота автоматизації доступу до розкладу занять необхідно підібрати такі засоби, які дозволяють реалізувати діалоговий інтерфейс, обробку команд користувача, бізнес-логіку, роботу з базою даних, імпорт офіційного розкладу та механізм нагадувань.

Під час вибору інструментарію враховувалися такі критерії: доступність документації, поширеність технології, простота інтеграції між окремими частинами системи, безкоштовність або доступність для навчального використання, а також відповідність темі дипломного проєкту. Оскільки система реалізується у форматі Telegram-бота, основою взаємодії з користувачем обрано платформу Telegram і Telegram Bot API.

Telegram Bot API використовується для обміну повідомленнями між користувачем і програмною системою. Саме через нього бот отримує команди, текстові повідомлення та натискання на кнопки, а також надсилає сформовані відповіді. Перевагою Telegram Bot API є те, що він дозволяє створювати програмні системи з готовим інтерфейсом взаємодії без потреби розробляти окремий вебсайт чи мобільний застосунок. [3]

Основною мовою програмування обрано Python. Вона є однією з найпоширеніших мов для розробки Telegram-ботів, має зрозумілий синтаксис, велику кількість бібліотек і добре підходить для створення прикладних систем навчального характеру. Python є доцільним вибором для цього проєкту, оскільки дозволяє поєднати обробку команд, роботу з файлами Excel, бізнес-логіку та взаємодію з базою даних у межах однієї технологічної платформи. [1]

Для реалізації самого Telegram-бота використовується фреймворк aiogram. Він забезпечує зручну роботу з Telegram Bot API, підтримує асинхронну обробку подій, дозволяє будувати модульну структуру застосунку та розділяти обробку команд, логіку меню й внутрішні сервіси. Це особливо важливо для даної системи, оскільки бот має кілька функціональних блоків: перегляд розкладу, роботу з вибірковими дисциплінами, профілем і нагадуваннями. [2]

Для побудови інтерфейсу користувача в межах Telegram використовується ReplyKeyboardMarkup та інші засоби формування клавіатур і кнопок. Це дозволяє створити просте й зрозуміле меню, через яке студент може швидко перейти до потрібного розділу: перегляду розкладу, профілю, вибіркових дисциплін чи нагадувань. Такий підхід є зручним, оскільки користувачеві не потрібно вводити команди вручну. [2; 3]

Для зчитування та обробки офіційного Excel-файлу розкладу використовується бібліотека openpyxl. Вона дозволяє відкривати файли формату .xlsx, працювати зі структурами електронних таблиць, отримувати значення комірок та обробляти розклад у вигляді, придатному для подальшого збереження в базі даних. Використання openpyxl є обґрунтованим, оскільки офіційний розклад публікується саме у форматі Excel. [5; 6]

Для збереження даних у системі обрано SQLite. Це вбудована реляційна система управління базами даних, яка не потребує встановлення окремого серверного програмного забезпечення та добре інтегрується з Python. SQLite є доцільною для навчального проєкту, оскільки забезпечує підтримку реляційної структури, зовнішніх ключів, просте резервне копіювання та зручність перенесення системи між різними середовищами. [4]

Для роботи з датою та часом використовуються стандартні засоби Python, зокрема модулі datetime та пов’язана з ними логіка. Вони застосовуються для визначення поточного типу навчального тижня, формування дат нагадувань, фіксації часу реєстрації користувачів і часу імпорту розкладу. Це дозволяє реалізувати часову логіку системи без залучення зайвих зовнішніх залежностей. [1]

Для реалізації нагадувань використовується окремий модуль scheduler, який відповідає за запуск перевірок у визначений момент часу та надсилання повідомлень користувачеві. Виділення нагадувань в окремий модуль є доцільним, оскільки ця частина системи працює незалежно від безпосереднього запиту користувача і потребує окремої логіки обробки часу та станів.

Під час розробки також використовуються Visual Studio Code як середовище програмування, Git для контролю версій та стандартні засоби налагодження Python. Це дозволяє зручно редагувати код, перевіряти працездатність системи, виявляти помилки та супроводжувати проєкт у процесі його розвитку. [8; 10]

Отже, для створення прикладного програмного забезпечення було обрано такий інструментарій: Telegram Bot API як основу взаємодії з користувачем; Python як мову програмування; aiogram як фреймворк для реалізації Telegram-бота; openpyxl для роботи з Excel-файлом розкладу; SQLite для збереження даних; datetime для роботи з датою та часом; scheduler для реалізації нагадувань; Visual Studio Code та Git як допоміжні засоби розробки.

Обраний набір технологій є доцільним для дипломного проєкту, оскільки дозволяє створити повноцінний Telegram-бот без надмірного ускладнення архітектури. Система залишається зрозумілою, гнучкою та придатною для подальшого розвитку: у майбутньому до неї можна додати підтримку кількох академічних груп, посилання на Google Meet, інтеграцію з eLearn, журнал змін розкладу або розширену систему сповіщень.

3.5. Алгоритмізація та програмування програмних модулів

Алгоритмізація програмних модулів є етапом, на якому загальна архітектура системи перетворюється на послідовність конкретних дій, що виконуються окремими частинами програмного забезпечення. Для Telegram-бота автоматизації доступу до розкладу занять основними модулями є модуль запуску бота, модуль головного меню, модуль перегляду розкладу, модуль вибіркових дисциплін, модуль профілю користувача, модуль нагадувань, модуль імпорту розкладу та модуль роботи з базою даних. Кожен із них виконує окрему функцію, але разом вони забезпечують повний сценарій роботи системи — від запуску бота до отримання персоналізованого розкладу.

Серверна частина системи реалізується на основі Python та фреймворку aiogram. Її алгоритм можна описати як послідовність запуску програми, підключення необхідних модулів, налаштування конфігурації, ініціалізації бази даних та реєстрації обробників команд. Після запуску бот переходить у режим очікування вхідних повідомлень і подій від користувача. Коли користувач надсилає команду або натискає кнопку меню, відповідний обробник отримує запит і передає його до модулів бізнес-логіки.

Алгоритм роботи головного модуля починається з запуску файлу bot.py. На цьому етапі виконується ініціалізація конфігурації, підключення до бази даних, реєстрація обробників команд і запуск циклу отримання повідомлень. Після цього система очікує взаємодії користувача з Telegram-ботом. Такий підхід дозволяє централізовано організувати роботу всіх інших модулів системи.

Одним із ключових є модуль головного меню. Його завдання — забезпечити перший контакт користувача із системою та надати доступ до основних функцій. Після запуску бота користувач отримує вітальне повідомлення та клавіатуру з основними пунктами меню. Алгоритм роботи цього модуля передбачає формування кнопок, обробку натискань і перенаправлення користувача до потрібного функціонального блоку: перегляду розкладу, профілю, вибіркових дисциплін або нагадувань.

Центральним для системи є модуль перегляду розкладу. Його алгоритм починається з того, що користувач обирає відповідний пункт меню та вказує режим перегляду: на сьогодні, завтра або тиждень. Після цього система отримує ідентифікатор користувача, зчитує його профіль, вибіркові дисципліни та визначає поточний тип навчального тижня. Далі бот звертається до бази даних, відбирає записи розкладу, фільтрує їх за групою, типом тижня та вибірковими дисциплінами, а потім формує текстову відповідь. Якщо для певного заняття передбачено додаткове посилання, наприклад на Google Meet або eLearn, воно також додається до повідомлення.

Окремо реалізується алгоритм визначення типу заняття. Під час імпорту Excel-файлу система аналізує структуру записів і визначає, чи є заняття лекційним або практичним. Якщо запис про дисципліну є спільним для кількох академічних груп, він трактується як лекція. Якщо ж запис подано окремо для кожної групи, система визначає його як практичне заняття. Такий підхід дозволяє зробити розклад більш інформативним і в подальшому використовувати ознаку типу заняття для розширення функціональності.

Модуль вибіркових дисциплін забезпечує перегляд і зміну індивідуального набору дисциплін користувача. Алгоритм його роботи включає отримання списку доступних вибіркових дисциплін, виведення цього списку користувачу, прийняття введених даних, перевірку їх коректності та оновлення відповідних записів у базі даних. Після збереження нових значень система підтверджує успішне оновлення вибору. Саме цей модуль забезпечує персоналізацію розкладу.

Модуль профілю користувача відповідає за відображення індивідуальних параметрів студента. Після звернення до цього розділу бот отримує профіль із бази даних та показує користувачу пов’язану інформацію, зокрема групу, поточний тип тижня і стан нагадувань. За потреби окремі параметри можуть бути оновлені або використані в інших частинах системи.

Модуль нагадувань працює паралельно з основними сценаріями взаємодії. Його алгоритм полягає у перевірці часу надсилання повідомлень, отриманні потрібних даних із бази та формуванні тексту нагадування. Якщо час нагадування настав, система створює повідомлення про майбутні заняття та надсилає його користувачу. У межах цього модуля також реалізується вмикання та вимикання нагадувань, зміна їх параметрів і збереження відповідних налаштувань у базі даних.

Модуль імпорту розкладу відповідає за зчитування офіційного Excel-файлу. Його алгоритм складається з відкриття файлу, пошуку потрібного фрагмента розкладу, обробки рядків і комірок, нормалізації назв дисциплін, визначення типу заняття та запису отриманих даних у базу. Під час імпорту також враховується тип тижня, номер пари, аудиторія або формат проведення та ознака вибірковості. Саме цей модуль забезпечує актуальність інформації, якою надалі користується бот.

Модуль роботи з базою даних забезпечує збереження і читання всіх основних сутностей системи. Його алгоритми охоплюють створення користувачів, оновлення профілів, запис вибіркових дисциплін, збереження нагадувань та отримання записів розкладу. База даних виступає центральним сховищем, що забезпечує цілісність інформації та можливість повторного використання вже збережених даних у різних сценаріях роботи.

Під час програмування модулів важливо дотримуватися принципу розділення відповідальності. Модулі обробки команд не повинні містити всю бізнес-логіку, а модулі бізнес-логіки не повинні безпосередньо формувати елементи інтерфейсу. Таке розділення дозволяє зробити код більш зрозумілим, спростити супровід системи та зменшити залежність між окремими частинами програми.

У процесі реалізації також потрібно враховувати помилки та граничні ситуації. Наприклад, користувач може ввести некоректний номер вибіркової дисципліни, розклад на обраний день може бути відсутнім, під час імпорту Excel-файлу можливі пропуски або неочікувана структура таблиці, а в базі даних

можуть бути відсутні деякі записи профілю. Для таких ситуацій програмні модулі повинні передбачати перевірки й коректні повідомлення користувачу.

Алгоритмізація та програмування модулів забезпечують практичну реалізацію архітектури системи. Основними алгоритмами є запуск бота, обробка меню, формування персоналізованого розкладу, визначення типу навчального тижня, робота з вибірковими дисциплінами, імпорт офіційного розкладу, надсилання нагадувань та взаємодія з базою даних. Сукупність цих модулів формує працездатне програмне забезпечення для автоматизації доступу до розкладу занять студентів.

РОЗДІЛ 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1. Тестування системи

Тестування системи є важливим етапом розробки програмного забезпечення, оскільки дозволяє перевірити правильність роботи основних функцій, виявити помилки та оцінити готовність програмного продукту до демонстрації й подальшої експлуатації. Для Telegram-бота автоматизації доступу до розкладу занять тестування має охоплювати не лише перевірку інтерфейсу, а й роботу з розкладом, вибірковими дисциплінами, нагадуваннями, імпортом Excel-файлу та збереженням даних у базі. [13; 15]

Метою тестування є підтвердження того, що система виконує основні вимоги, сформульовані в попередніх розділах: дозволяє запускати Telegram-бота, переглядати розклад на сьогодні, завтра та тиждень, враховувати вибіркові дисципліни користувача, автоматично визначати тип навчального тижня, працювати з профілем користувача та підтримувати нагадування про заняття.

Тестування проводиться у кілька етапів. Спочатку перевіряється запуск бота та доступність головного меню. Під час цього етапу необхідно впевнитися, що після команди запуску бот коректно надсилає вітальне повідомлення та відображає основні кнопки навігації. Результат перевірки запуску бота та відображення головного меню наведено на рис. 11.

Рис. 11 – Запуск Telegram-бота та відображення головного меню

Після цього тестуються основні сценарії взаємодії користувача із системою, зокрема відкриття розділу розкладу та перегляд занять на сьогодні. На цьому етапі необхідно перевірити, що бот коректно зчитує дані користувача, визначає поточний день і формує актуальний перелік занять. Приклад перевірки формування розкладу на сьогодні наведено на рис. 12.

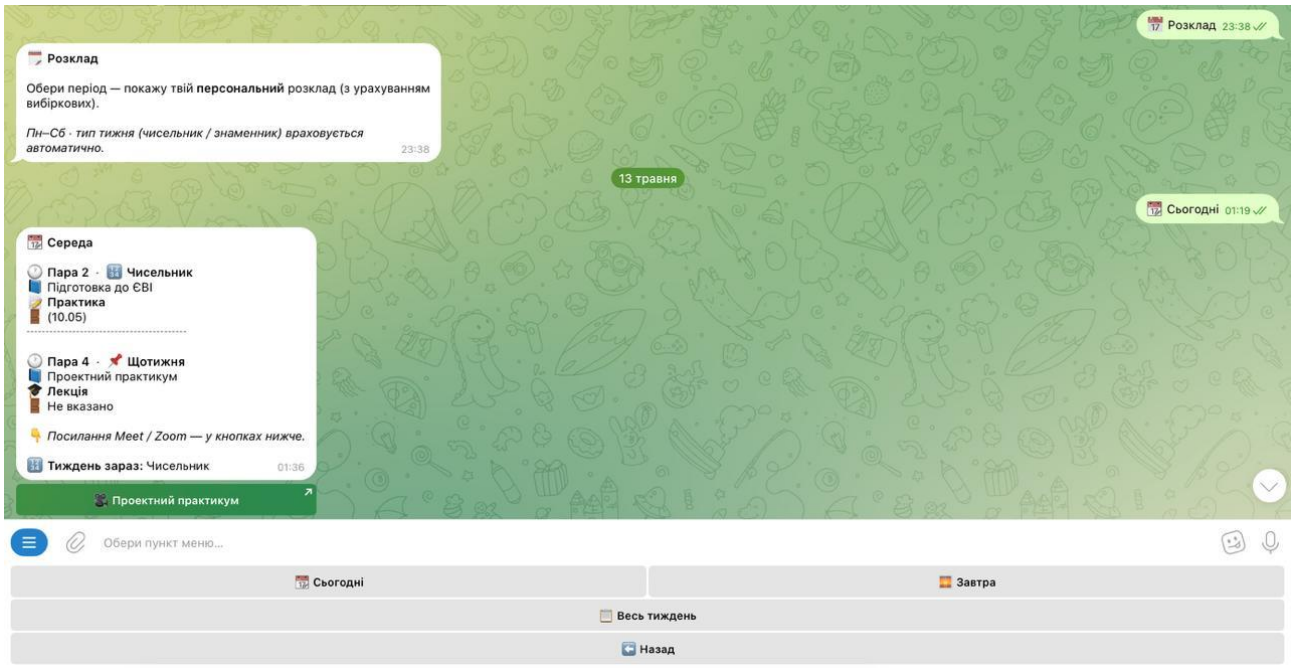

Рис. 12 – Результат перевірки формування розкладу на сьогодні

Окремо перевіряється формування розкладу на інші часові періоди, зокрема на завтра або на тиждень. Це дозволяє оцінити коректність вибору записів за датою та типом навчального тижня. Результат перевірки формування розкладу на інший часовий період наведено на рис. 13.

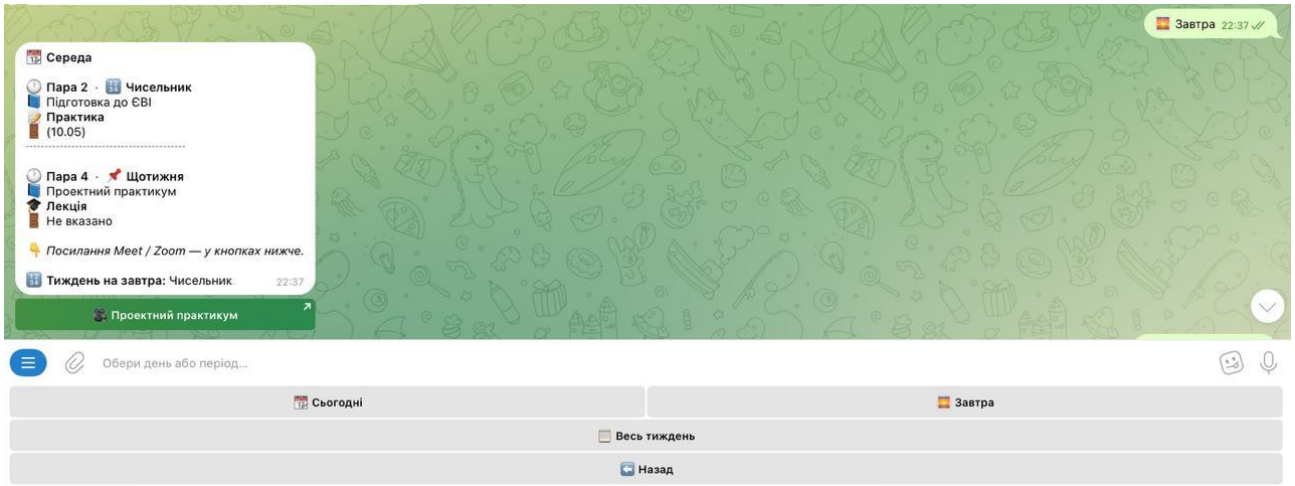

Рис. 13 – Результат перевірки формування розкладу на завтра

Окремо тестується модуль формування персоналізованого розкладу, оскільки саме він є центральним у роботі системи. Під час перевірки необхідно впевнитися, що бот коректно зчитує дані користувача з бази, визначає поточний тип навчального тижня, отримує записи розкладу й правильно застосовує вибіркові дисципліни. Якщо хоча б один із цих елементів працює неправильно, користувач може отримати неповний або некоректний розклад.

Функції роботи з вибірковими дисциплінами перевіряються окремо. Необхідно переконатися, що користувач може переглядати список власних вибіркових предметів, змінювати їх, а результати змін зберігаються в базі даних. Після зміни вибору бот повинен формувати оновлений персоналізований розклад без помилок і дублювання записів. Перевірку роботи модуля вибіркових дисциплін наведено на рис. 14.

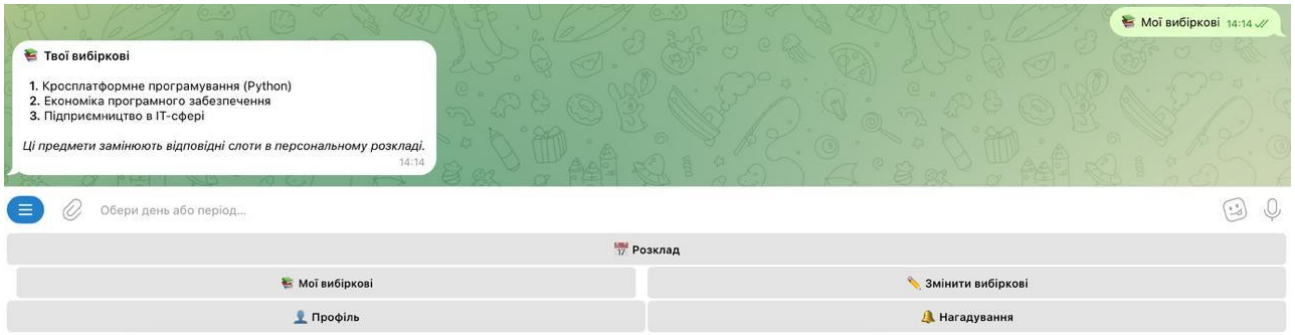

Рис. 14 – Перевірка перегляду та зміни вибіркових дисциплін

Окремим модулем тестується система нагадувань. Потрібно перевірити, чи зберігається стан нагадувань у профілі користувача, чи формуються записи в таблиці нагадувань і чи надсилає бот повідомлення у потрібний час. Якщо для заняття передбачено додаткове посилання, наприклад на Google Meet, необхідно переконатися, що воно також коректно додається до повідомлення. Результат перевірки роботи системи нагадувань наведено на рис. 15.

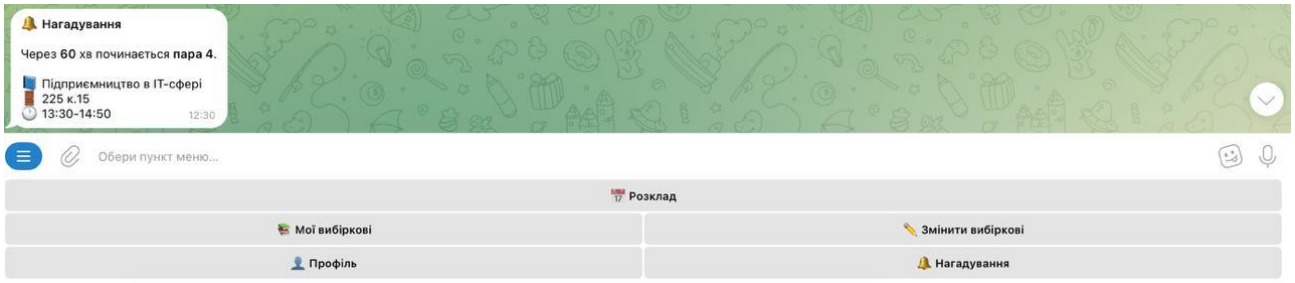

Рис. 15 – Результат перевірки роботи системи нагадувань

Також необхідно перевірити поведінку системи в граничних і помилкових ситуаціях. До таких ситуацій належать: відсутність занять на вибраний день, неправильне введення даних користувачем, відсутність записів у профілі, некоректна структура Excel-файлу, дублювання вибіркових дисциплін або відсутність доступу до деяких зовнішніх даних. Система повинна або коректно обробляти такі ситуації, або принаймні не завершувати роботу аварійно та повідомляти користувача про проблему. Приклад обробки помилкової або граничної ситуації наведено на рис. 16.

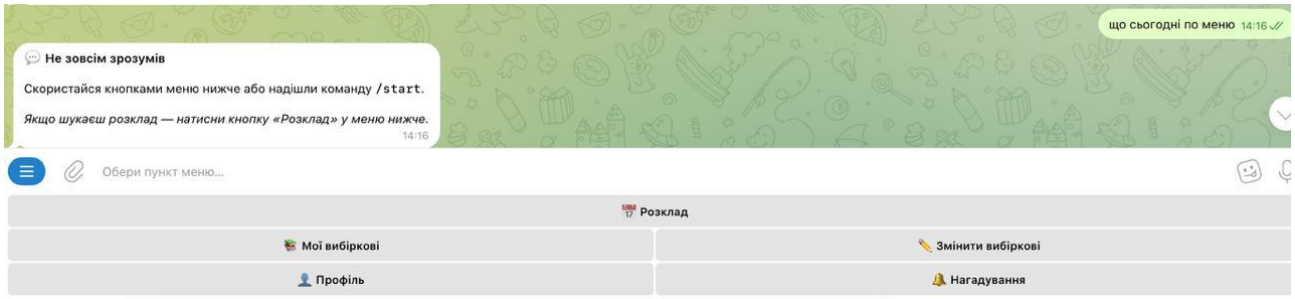

Рис. 16 – Приклад обробки помилкової або граничної ситуації

№	Що перевірялось	Очікуваний результат	Фактичний результат
1	Запуск бота	Відображається вітальне повідомлення та головне меню	Успішно
2	Перегляд розкладу на сьогодні	Бот формує актуальний перелік занять	Успішно
3	Перегляд розкладу на завтра/тиждень	Бот відображає відповідні записи розкладу	Успішно
4	Перегляд і зміна вибіркових дисциплін	Вибір зберігається в базі даних	Успішно
5	Імпорт Excel-файлу	Дані коректно зчитуються та записуються в БД	Успішно
6	Робота нагадувань	Користувач отримує повідомлення у заданий час	Успішно
7	Відображення профілю	Бот показує збережені параметри користувача	Успішно
8	Помилкова ситуація	Система коректно повідомляє про проблему	Успішно

Результати тестування показують, чи відповідає система основним вимогам і чи може вона використовуватись для демонстрації та практичного застосування. Якщо виявлено обмеження, їх необхідно зазначити як напрями подальшого вдосконалення. Наприклад, у майбутньому можна розширити механізм нагадувань, додати аналіз змін у розкладі, покращити імпорт даних або реалізувати підтримку кількох академічних груп.

Тестування системи підтверджує працездатність основних модулів: запуску бота, меню, перегляду розкладу, роботи з вибірковими дисциплінами, профілю, нагадувань, імпорту Excel-файлу та взаємодії з базою даних. Проведення таких перевірок дозволяє оцінити якість реалізації, виявити слабкі місця та підготувати систему до впровадження й подальшої експлуатації.

4.2. Вимоги до апаратного та програмного забезпечення

Оскільки розроблювана система є Telegram-ботом, основні вимоги поділяються на дві групи: вимоги до серверного середовища, де запускається бот, та вимоги до клієнтського пристрою, з якого користувач взаємодіє із системою через Telegram. У межах цієї роботи сервер виконує обробку команд, формування відповідей, імпорт розкладу, роботу з нагадуваннями та взаємодію з базою даних. Тому вимоги до серверної частини є помірними й не потребують високопродуктивної інфраструктури. [1; 2; 3; 4]

Для запуску серверної частини необхідний персональний комп’ютер або віртуальний сервер із встановленим середовищем Python 3. Мінімальними апаратними характеристиками для демонстраційного або навчального запуску можна вважати: процесор із 2 ядрами, 2 ГБ оперативної пам’яті, 1–2 ГБ вільного місця на диску для файлів проєкту, залежностей та бази даних, а також стабільне підключення до мережі Інтернет. Для більш комфортної роботи бажано використовувати процесор із 4 ядрами, 4 ГБ оперативної пам’яті та швидке мережеве з’єднання. Якщо кількість одночасних користувачів зростатиме, вимоги до серверного середовища також збільшуватимуться, насамперед через кількість запитів до Telegram Bot API, роботу модуля нагадувань та обсяг операцій із базою даних.

Оскільки система не обробляє складні мультимедійні потоки, а працює переважно з текстовими повідомленнями, файлами Excel та записами бази даних, сервер не потребує значних апаратних ресурсів. Це є перевагою обраної архітектури. Водночас сервер повинен мати стабільний доступ до Інтернету, оскільки через нього здійснюється обмін даними з Telegram Bot API та, за потреби, отримання інформації із зовнішнього джерела розкладу. Для демонстраційного режиму достатньо локального запуску на персональному комп’ютері, але для постійної експлуатації доцільно використовувати VPS або хмарний сервер із цілодобовим доступом до мережі.

До програмного забезпечення серверного середовища належать: операційна система Windows, Linux або macOS, середовище виконання Python 3,

встановлені бібліотеки проєкту, зокрема aiogram, openpyxl, sqlite3 та інші залежності, визначені у файлі requirements.txt. Також необхідно налаштувати конфігураційні параметри, зокрема токен Telegram-бота, службові налаштування та параметри доступу до файлів і бази даних. Для розробки та супроводу системи доцільно використовувати середовище Visual Studio Code та систему контролю версій Git. [1; 2; 4; 10]

Для роботи з базою даних окремий фізичний сервер у межах цієї системи не є обов'язковим, оскільки використовується SQLite. Вона зберігає дані у вигляді локального файлу та не потребує окремого сервісу баз даних. Це спрощує розгортання системи, резервне копіювання та перенесення проєкту між різними середовищами. [4]

Для клієнтської частини потрібен сучасний пристрій із встановленим додатком Telegram або доступом до Telegram Web та стабільним підключенням до Інтернету. Це може бути смартфон, планшет, ноутбук або персональний комп'ютер. Для повноцінної роботи достатньо, щоб користувач мав змогу надсилати повідомлення, користуватися меню та отримувати відповіді від бота. Оскільки система не потребує камери, мікрофона чи спеціальних мультимедійних пристроїв, вимоги до клієнтського обладнання є нижчими порівняно з системами відеозв'язку. [3; 23]

Якість роботи клієнтської частини значною мірою залежить від стабільності мережі. При слабкому інтернет-з'єднанні можливі затримки отримання повідомлень або повільне завантаження оновлених даних, однак сама система залишається працездатною, оскільки обмін інформацією відбувається переважно у текстовому форматі. Для щоденного використання достатньо стандартного мобільного або домашнього інтернет-з'єднання.

Для демонстрації системи не потрібне складне зовнішнє розгортання, оскільки користувач взаємодіє з ботом через Telegram. Якщо ж бот має працювати постійно, доцільно розміщувати його на сервері або VPS із постійним доступом до Інтернету. Це забезпечить безперервну роботу системи, своєчасне

надсилання нагадувань і доступність функцій для користувачів у будь-який момент часу.

Отже, вимоги до апаратного та програмного забезпечення розроблюваної системи є помірними. Для серверної частини достатньо комп’ютера або віртуального сервера з Python 3, доступом до Інтернету та встановленими залежностями проєкту. Для клієнтської частини потрібен будь-який сучасний пристрій із Telegram і стабільним мережевим з’єднанням. Такий підхід робить систему доступною, зручною в розгортанні та придатною як для демонстрації, так і для подальшої експлуатації.

4.3. Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення призначений для передавання, розгортання та запуску Telegram-бота на локальному комп'ютері або сервері. Оскільки розроблювана система реалізується на основі Python, інсталяційний пакет має містити вихідний код програми, файли конфігурації, опис залежностей, базу даних, допоміжні файли для імпорту розкладу та інструкції щодо запуску системи. [1; 2; 4; 5]

До складу інсталяційного пакету входить основна програмна частина системи. Вона містить головний файл запуску bot.py, який відповідає за ініціалізацію Telegram-бота, підключення до бази даних, реєстрацію обробників команд та запуск системи. Саме цей файл забезпечує початок роботи програмного забезпечення та координує взаємодію між окремими модулями.

Окремою частиною пакету є модулі програмного забезпечення, які реалізують окремі функціональні блоки системи. До них належать модулі handlers, що відповідають за обробку команд користувача, модулі services або logic, у яких реалізується бізнес-логіка, модуль database.py для роботи з базою даних, модуль scheduler.py для формування нагадувань, модуль parser.py для імпорту офіційного розкладу, а також файли клавіатур і конфігурації. Такий поділ дозволяє зробити структуру проєкту зрозумілою та придатною до супроводу.

Важливим елементом інсталяційного пакету є файл requirements.txt. У ньому зазначається перелік залежностей, необхідних для запуску системи, зокрема aiogram, openpyxl та інші використані бібліотеки. Після отримання інсталяційного пакету користувач або адміністратор повинен встановити залежності за допомогою команди pip install -r requirements.txt. [1; 2; 5]

До складу пакету також входить файл конфігурації, наприклад .env.example або окремий конфігураційний файл, у якому задаються службові параметри: токен Telegram-бота, шляхи до файлів, параметри середовища та інші

налаштування. Реальний файл .env, якщо він містить конфіденційні дані, не повинен передаватися разом із відкритим кодом без необхідності.

Для роботи з інформаційною базою доцільно включити файл локальної бази даних SQLite або окрему інструкцію щодо її створення. Якщо база даних не передається у готовому вигляді, потрібно додати SQL-скрипти або код ініціалізації таблиць, що дозволяє швидко підготувати структуру бази даних до роботи. До такої структури входять таблиці користувачів, профілів, вибіркових дисциплін, вибору користувача, записів розкладу та нагадувань. [4]

Окремим елементом інсталяційного пакету є офіційний Excel-файл розкладу або інструкція щодо його завантаження та підключення до системи. Оскільки бот використовує цей файл як джерело даних, його наявність є важливою умовою для коректної роботи програми. За потреби разом із пакетом можуть постачатися допоміжні файли для тестового запуску або приклади вхідних даних. [5; 6]

Також до інсталяційного пакету варто включити інструкцію користувача або адміністратора. У ній необхідно описати порядок встановлення Python, створення віртуального середовища, встановлення залежностей, налаштування конфігураційних файлів, запуск Telegram-бота та перевірку основних функцій. Якщо система запускається на сервері, до інструкції доцільно додати рекомендації щодо постійного запуску бота та забезпечення доступу до Інтернету.

До додаткових матеріалів можуть входити діаграми, фрагменти коду, результати тестування, приклади повідомлень бота та інша технічна документація. Вони не є обов’язковими для запуску системи, але корисні для пояснення архітектури, захисту дипломного проєкту та подальшого супроводу програмного забезпечення.

Після розпакування інсталяційного пакету користувач або адміністратор має встановити залежності, налаштувати конфігураційні параметри, підготувати базу даних і, за потреби, імпортувати розклад із Excel-файлу. Після цього бот

можна запустити, і він стане доступним у середовищі Telegram за відповідним іменем користувача.

Інсталяційний пакет має містити всі необхідні файли для запуску й перевірки програмного забезпечення: вихідний код бота, модулі системи, опис залежностей, конфігураційні файли, базу даних або засоби її створення, файл розкладу та інструкцію з розгортання. Такий склад забезпечує можливість відтворити роботу системи на іншому комп’ютері або сервері без додаткового пошуку потрібних компонентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Software Foundation. Python Documentation. URL: https://docs.python.org/3/
2. aiogram Team. aiogram Documentation. URL: https://docs.aiogram.dev/
3. Telegram. Telegram Bot API. URL: https://core.telegram.org/bots/api
4. SQLite. SQLite Documentation. URL: https://sqlite.org/docs.html
5. openpyxl. openpyxl documentation. URL: https://openpyxl.readthedocs.io/
6. Microsoft Learn. Structure of a SpreadsheetML document. URL: https://learn.microsoft.com/en-us/office/open-xml/spreadsheet/structure-of-a-spreadsheetml-document
7. W3C. Extensible Markup Language (XML). URL: https://www.w3.org/XML/
8. Git. Git Documentation. URL: https://git-scm.com/doc
9. GitHub Docs. About GitHub and Git. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git
10. Microsoft. Documentation for Visual Studio Code. URL: https://code.visualstudio.com/docs
11. Microsoft. Python extension for Visual Studio Code. URL: https://marketplace.visualstudio.com/items?itemName=ms-python.python
12. OWASP Foundation. OWASP Top 10 Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/
13. OWASP Foundation. OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/
14. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.
15. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill Education, 2020. 672 p.
16. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 208 p.

17. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston : Addison-Wesley, 2005. 496 p.

18. Google. Google Meet Help. URL: https://support.google.com/meet/

19. Telegram. Telegram Applications. URL: https://telegram.org/apps

20. NIST. Digital Identity Guidelines (SP 800-63-4). URL: https://pages.nist.gov/800-63-4/

21. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/

ДОДАТОК А

ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

У додатку А наведено фрагменти програмного коду, розробленого для реалізації Telegram-бота автоматизації доступу до персоналізованого розкладу занять студентів. Подані фрагменти відображають основні функціональні частини системи: запуск бота, обробку команд користувача, роботу з розкладом, вибірковими дисциплінами, нагадуваннями, базою даних та імпортом офіційного Excel-файлу розкладу. Наведені матеріали підтверджують практичну реалізацію запропонованої архітектури програмного забезпечення.

А.1. Фрагмент коду запуску Telegram-бота

У даному підпункті наведено фрагмент коду головного файлу bot.py, який відповідає за ініціалізацію Telegram-бота, підключення модулів системи, запуск імпорту розкладу з Excel-файлу, перевірку з’єднання з Telegram API, запуск модуля нагадувань та переведення системи в режим очікування вхідних повідомлень.

```
import asyncio
import logging
import os
import sys
from pathlib import Path

from app.config import BOT_TOKEN
from app.database.db import init_db

logger = logging.getLogger(__name__)


async def main() -> None:
    from aiogram import Bot, Dispatcher
    from app.handlers import electives, fallback, reminders, schedule, start
    from app.services.excel_importer import import_group_schedule_with_report
    from app.services.reminder_engine import reminder_tick

    init_db()

    excel_path = Path("data/faculty_schedule.xlsx")
    skip_excel = os.getenv("IMPORT_SCHEDULE_ON_START", "1").lower() in (
        "0",
        "false",
        "no",
    )
    if skip_excel:
        print(
            "IMPORT_SCHEDULE_ON_START=0 — імпорт Excel на старті вимкнено.",
            flush=True,
        )
```

```
elif not excel_path.is_file():
    print(
        f"Файл {excel_path.resolve()} відсутній — імпорт розкладу пропущено.",
        flush=True,
    )
else:
    try:
        import_report = import_group_schedule_with_report(
            excel_path, dry_run=False
        )
        print(
            "Schedule import:",
            import_report["parsed_items"],
            "items;",
            len(import_report["warnings"]),
            "warnings",
            flush=True,
        )
    except Exception as exc:
        logger.exception("Імпорт розкладу з Excel не вдався")
        print(
            f"Помилка імпорту розкладу: {exc}\n"
            "Бот запускається з даними, що вже є в БД.",
            flush=True,
        )

bot = Bot(token=BOT_TOKEN)
dp = Dispatcher()

dp.include_router(start.router)
dp.include_router(schedule.router)
dp.include_router(reminders.router)
dp.include_router(electives.router)
dp.include_router(fallback.router)

print("Підключення до Telegram API…", flush=True)
try:
    me = await asyncio.wait_for(bot.get_me(), timeout=45.0)
except asyncio.TimeoutError:
    print(
        "Не вдалося з'єднатися з api.telegram.org за 45 с.\n"
```

```
            "Перевір інтернет, VPN, файрвол і чи не блокується Telegram.",
            file=sys.stderr,
            flush=True,
        )
        raise SystemExit(1) from None
    except Exception as exc:
        print(f"Помилка Telegram (перевір BOT_TOKEN у .env): {exc}", file=sys.stderr,
flush=True)
        raise SystemExit(1) from exc

    await bot.delete_webhook(drop_pending_updates=True)
    un = f"@{me.username}" if me.username else str(me.id)
    print(f"OK: бот {un}. Webhook скинуто, long polling…", flush=True)

    async def _reminder_loop() -> None:
        from app.config import REMINDER_POLL_INTERVAL_SEC

        while True:
            try:
                await reminder_tick(bot)
            except Exception:
                logger.exception("reminder_tick")
            await asyncio.sleep(REMINDER_POLL_INTERVAL_SEC)

    asyncio.create_task(_reminder_loop())

    await dp.start_polling(bot)


if __name__ == "__main__":
    logging.basicConfig(
        level=logging.INFO,
        format="%(levelname)s %(name)s: %(message)s",
    )
    print("Старт бота…", flush=True)
    asyncio.run(main())
```

А.2. Фрагменти коду модуля роботи з базою даних

У даному підпункті наведено фрагменти коду модуля app/database/db.py, який забезпечує підключення до бази даних SQLite, створення таблиць і виконання основних операцій збереження та отримання даних користувачів, вибіркових дисциплін, нагадувань і записів розкладу.

А.2.1. Фрагмент коду ініціалізації бази даних

У даному фрагменті наведено частину коду, яка відповідає за підключення до бази даних SQLite та створення основних таблиць системи.

```
import sqlite3
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent.parent
DB_PATH = BASE_DIR / "schedule_bot.db"


def get_connection() -> sqlite3.Connection:
    connection = sqlite3.connect(DB_PATH, timeout=15.0)
    connection.row_factory = sqlite3.Row
    return connection


def init_db() -> None:
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            telegram_id INTEGER NOT NULL UNIQUE,
            username TEXT,
            first_name TEXT,
            last_name TEXT,
            is_active INTEGER NOT NULL DEFAULT 1,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
    )
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS user_profiles (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL UNIQUE,
        schedule_group_name TEXT NOT NULL,
        display_name TEXT,
        course_year TEXT,
        notes TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY(user_id) REFERENCES users(id)
    )
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS elective_subjects (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL UNIQUE,
        code TEXT,
        description TEXT,
        is_active INTEGER NOT NULL DEFAULT 1,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS user_elective_subjects (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        subject_id INTEGER NOT NULL,
        assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        UNIQUE(user_id, subject_id),
        FOREIGN KEY(user_id) REFERENCES users(id),
        FOREIGN KEY(subject_id) REFERENCES elective_subjects(id)
    )
""")

cursor.execute("""
```

```
    CREATE TABLE IF NOT EXISTS reminders (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        reminder_kind TEXT NOT NULL DEFAULT 'before_pair',
        minutes_before INTEGER NOT NULL DEFAULT 15,
        is_enabled INTEGER NOT NULL DEFAULT 1,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY(user_id) REFERENCES users(id)
    )
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS schedule_items (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        group_name TEXT NOT NULL,
        day_of_week INTEGER NOT NULL,
        pair_number INTEGER NOT NULL,
        time_range TEXT,
        week_type TEXT NOT NULL,
        raw_subject_name TEXT NOT NULL,
        normalized_subject_name TEXT NOT NULL,
        room TEXT,
        teacher TEXT,
        is_elective INTEGER NOT NULL DEFAULT 0,
        lesson_type TEXT NOT NULL DEFAULT 'practice',
        meet_url TEXT,
        elearn_url TEXT,
        imported_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
""")

connection.commit()
connection.close()
```

А.2.2. Фрагмент коду роботи з користувачами та профілем

У даному фрагменті наведено частину коду, яка реалізує створення користувача, оновлення його даних та автоматичне створення профілю користувача.

```
from app.config import SCHEDULE_GROUP_NAME


def add_user(
    telegram_id: int,
    username: str | None,
    first_name: str | None,
    last_name: str | None
) -> None:
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        INSERT OR IGNORE INTO users (telegram_id, username, first_name, last_name)
        VALUES (?, ?, ?, ?)
    """, (telegram_id, username, first_name, last_name))

    cursor.execute("""
        UPDATE users
        SET username = ?, first_name = ?, last_name = ?,
            updated_at = CURRENT_TIMESTAMP
        WHERE telegram_id = ?
    """, (username, first_name, last_name, telegram_id))

    cursor.execute("SELECT id FROM users WHERE telegram_id = ?", (telegram_id,))
    row = cursor.fetchone()
    if row:
        cursor.execute("""
            INSERT OR IGNORE INTO user_profiles (user_id, schedule_group_name)
            VALUES (?, ?)
        """, (row["id"], SCHEDULE_GROUP_NAME))

        cursor.execute("""
            INSERT OR IGNORE INTO reminders (user_id, reminder_kind, minutes_before,
is_enabled)
```

```
            VALUES (?, 'before_pair', 15, 0)
        """, (row["id"],))

    connection.commit()
    connection.close()


def get_user_by_telegram_id(telegram_id: int):
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        SELECT * FROM users
        WHERE telegram_id = ?
    """, (telegram_id,))

    user = cursor.fetchone()
    connection.close()
    return user
```

А.2.3. Фрагмент коду роботи з вибірковими дисциплінами

У даному фрагменті наведено частину коду, яка забезпечує отримання списку вибіркових дисциплін, збереження вибору користувача та читання персональних налаштувань для формування індивідуального розкладу.

```
def get_all_elective_subjects():
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        SELECT * FROM elective_subjects
        WHERE COALESCE(is_active, 1) = 1
        ORDER BY id
    """)

    subjects = cursor.fetchall()
    connection.close()
    return subjects


def clear_user_elective_subjects(user_id: int) -> None:
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        DELETE FROM user_elective_subjects
        WHERE user_id = ?
    """, (user_id,))

    connection.commit()
    connection.close()


def add_user_elective_subject(user_id: int, subject_id: int) -> None:
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        INSERT OR IGNORE INTO user_elective_subjects (user_id, subject_id)
        VALUES (?, ?)
```

```
    """, (user_id, subject_id))

    connection.commit()
    connection.close()


def get_user_elective_subjects(user_id: int):
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        SELECT elective_subjects.id, elective_subjects.name
        FROM user_elective_subjects
        JOIN elective_subjects
            ON user_elective_subjects.subject_id = elective_subjects.id
        WHERE user_elective_subjects.user_id = ?
        ORDER BY elective_subjects.id
    """, (user_id,))

    subjects = cursor.fetchall()
    connection.close()
    return subjects
```

А.2.4. Фрагмент коду роботи з розкладом і нагадуваннями

У даному фрагменті наведено частину коду, яка реалізує збереження імпортованих записів розкладу, формування персоналізованого розкладу користувача та налаштування нагадувань.

```
def get_personal_schedule_for_day(user_id: int, day_of_week: int, current_week_type: str
= "numerator"):
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("""
        SELECT subject_id
        FROM user_elective_subjects
        WHERE user_id = ?
    """, (user_id,))
    user_subject_ids = [row["subject_id"] for row in cursor.fetchall()]

    cursor.execute("""
        SELECT name
        FROM elective_subjects
        WHERE id IN ({})
    """.format(",".join(["?"] * len(user_subject_ids))) if user_subject_ids else "SELECT
name FROM elective_subjects WHERE 1=0", user_subject_ids)

    selected_elective_names = {row["name"] for row in cursor.fetchall()}

    cursor.execute("""
        SELECT *
        FROM schedule_items
        WHERE group_name = ?
          AND day_of_week = ?
          AND (week_type = ? OR week_type = 'both')
          AND (record_status IS NULL OR record_status = 'active')
        ORDER BY pair_number, id
    """, (SCHEDULE_GROUP_NAME, day_of_week, current_week_type))

    all_items = cursor.fetchall()
    connection.close()

    base_items_by_slot = {}
```

```
elective_items_by_physical_slot = {}

for item in all_items:
    slot_key = item["slot_key"] or make_slot_key(
        day_of_week=item["day_of_week"],
        pair_number=item["pair_number"],
        week_type=item["week_type"],
        time_range=item["time_range"],
    )

    if item["is_elective"] == 0:
        base_items_by_slot[slot_key] = item
        continue

    if item["normalized_subject_name"] in selected_elective_names:
        physical_key = physical_schedule_slot_identity(item)
        elective_items_by_physical_slot[physical_key] = item

merged = []
processed_physical_slots = set()

for slot_key, base_item in sorted(
    base_items_by_slot.items(),
    key=lambda x: (x[1]["pair_number"], x[1]["id"]),
):
    physical_key = physical_schedule_slot_identity(base_item)
    if physical_key in elective_items_by_physical_slot:
        merged.append(elective_items_by_physical_slot[physical_key])
    else:
        merged.append(base_item)
    processed_physical_slots.add(physical_key)

for physical_key, elective_item in sorted(
    elective_items_by_physical_slot.items(),
    key=lambda x: (x[1]["pair_number"], x[1]["id"]),
):
    if physical_key not in processed_physical_slots:
        merged.append(elective_item)

return sorted(merged, key=lambda item: (item["pair_number"], item["id"]))
```

```
def get_reminder_settings(user_id: int):
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM reminders WHERE user_id = ?", (user_id,))
    row = cursor.fetchone()
    connection.close()
    return row


def set_reminder_enabled(user_id: int, enabled: bool) -> None:
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute("""
        UPDATE reminders
        SET is_enabled = ?, updated_at = CURRENT_TIMESTAMP
        WHERE user_id = ?
    """, (1 if enabled else 0, user_id))
    connection.commit()
    connection.close()
```

А.3. Фрагмент коду обробки команди запуску

У даному підпункті наведено фрагмент коду модуля app/handlers/start.py, який реалізує обробку команди запуску Telegram-бота, реєстрацію користувача в системі, формування вітального повідомлення та відображення головного меню.

```
import html

from aiogram import Router
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message

from app.config import SCHEDULE_GROUP_NAME
from app.database.db import add_user
from app.keyboards.main_menu import main_menu

router = Router()


@router.message(CommandStart())
async def start_handler(message: Message) -> None:
    user = message.from_user

    add_user(
        telegram_id=user.id,
        username=user.username,
        first_name=user.first_name,
        last_name=user.last_name,
    )

    name = html.escape(user.first_name or "друже")
    group = html.escape(SCHEDULE_GROUP_NAME)

    text = (
        f"Привіт, <b>{name}</b>! 👋\n\n"
        f"Я бот розкладу для групи <b>{group}</b>.\n\n"
        "ℹ️ <b>Що вмію</b>\n"
```

```
    "├ показувати розклад на сьогодні, завтра та весь тиждень\n"
    "├ враховувати твої <b>вибіркові</b> дисципліни\n"
    "├ показувати профіль і (згодом) нагадування\n"
    "└ давати швидкі посилання на Meet / eLearn, де вони налаштовані\n\n"
    "👇 <b>Обери дію</b> в меню нижче — кнопки завжди під рукою."
)

await message.answer(
    text,
    parse_mode=ParseMode.HTML,
    reply_markup=main_menu,
)
```

А.4. Фрагменти коду модуля перегляду розкладу

У даному підпункті наведено фрагменти коду модуля app/handlers/schedule.py, який реалізує взаємодію користувача з функціями перегляду персоналізованого розкладу на сьогодні, завтра та тиждень, а також забезпечує відображення профілю користувача. Модуль використовує дані з бази, враховує тип навчального тижня та формує повідомлення у форматі, зручному для відображення в Telegram.

А.4.1. Фрагмент коду меню розкладу та перегляду розкладу на сьогодні

У даному фрагменті наведено частину коду, яка відповідає за відкриття меню розкладу, перевірку наявності користувача в системі та формування персоналізованого розкладу на поточний день.

```
import html

from aiogram import Router
from aiogram.enums import ParseMode
from aiogram.types import LinkPreviewOptions, Message

from app.database.db import (
    get_personal_schedule_for_day,
    get_today_week_type,
    get_today_weekday,
    get_user_by_telegram_id,
)
from app.keyboards.schedule_menu import schedule_menu
from app.services.schedule_messages import (
    format_schedule,
    format_week_type_footer,
)
from app.ui_strings import MainKB, ScheduleKB

router = Router()

_PREVIEW_OFF = LinkPreviewOptions(is_disabled=True)
```

```
DAY_NAMES = {
    0: "Понеділок",
    1: "Вівторок",
    2: "Середа",
    3: "Четвер",
    4: "П’ятниця",
    5: "Субота",
    6: "Неділя",
}


def _not_registered_reply() -> str:
    return (
        "😕 <b>Користувача не знайдено</b>\n\n"
        "Натисни /start, щоб зареєструватися в боті."
    )


@router.message(lambda m: m.text == MainKB.SCHEDULE)
async def schedule_menu_handler(message: Message) -> None:
    await message.answer(
        "ℹ️ <b>Розклад</b>\n\n"
        "Обери період — покажу твій <b>персональний</b> розклад "
        "(з урахуванням вибіркових).\n\n"
        "<i>Пн–Сб · тип тижня (чисельник / знаменник) враховується автоматично.</i>",
        parse_mode=ParseMode.HTML,
        reply_markup=schedule_menu,
    )


@router.message(lambda m: m.text == ScheduleKB.TODAY)
async def today_handler(message: Message) -> None:
    user = get_user_by_telegram_id(message.from_user.id)

    if not user:
        await message.answer(
            _not_registered_reply(),
            parse_mode=ParseMode.HTML,
        )
        return
```

```
day = get_today_weekday()

if day > 5:
    await message.answer(
        "🌴 <b>Сьогодні вихідний</b>\n\n"
        "У розкладі зазвичай немає занять на неділю. Гарного відпочинку!",
        parse_mode=ParseMode.HTML,
    )
    return

week_type = get_today_week_type()
schedule = get_personal_schedule_for_day(user["id"], day, week_type)

body, meet_markup = format_schedule(
    schedule, DAY_NAMES[day], meet_inline_keyboard=True
)
footer = format_week_type_footer(week_type, context="today")
text = f"{body}\n\n{footer}"

await message.answer(
    text,
    parse_mode=ParseMode.HTML,
    reply_markup=meet_markup,
    link_preview_options=_PREVIEW_OFF,
)
```

А.4.2. Фрагмент коду перегляду розкладу на завтра

У даному фрагменті наведено частину коду, яка забезпечує формування персоналізованого розкладу на наступний день з урахуванням актуального типу навчального тижня.

```
@router.message(lambda m: m.text == ScheduleKB.TOMORROW)
async def tomorrow_handler(message: Message) -> None:
    user = get_user_by_telegram_id(message.from_user.id)

    if not user:
        await message.answer(
            _not_registered_reply(),
            parse_mode=ParseMode.HTML,
        )
        return
    day = get_tomorrow_weekday()

    if day > 5:
        await message.answer(
            "🌴 <b>Завтра вихідний</b>\n\n"
            "Занять у розкладі зазвичай немає. Можна планувати відпочинок 🙂",
            parse_mode=ParseMode.HTML,
        )
        return

    week_type = get_tomorrow_week_type()
    schedule = get_personal_schedule_for_day(user["id"], day, week_type)

    body, meet_markup = format_schedule(
        schedule, DAY_NAMES[day], meet_inline_keyboard=True
    )
    footer = format_week_type_footer(week_type, context="tomorrow")
    text = f"{body}\n\n{footer}"

    await message.answer(
        text,
        parse_mode=ParseMode.HTML,
        reply_markup=meet_markup,
        link_preview_options=_PREVIEW_OFF,
    )
```

А.4.3. Фрагмент коду формування тижневого розкладу

У даному фрагменті наведено частину коду, яка реалізує формування тижневого розкладу та розбиття великого повідомлення на кілька частин для коректного надсилання в Telegram.

```
@router.message(lambda m: m.text == ScheduleKB.WEEK)
async def week_handler(message: Message) -> None:
    user = get_user_by_telegram_id(message.from_user.id)

    if not user:
        await message.answer(
            _not_registered_reply(),
            parse_mode=ParseMode.HTML,
        )
        return

    week_type = get_today_week_type()
    week_schedule = get_personal_schedule_for_week(user["id"], week_type)

    header = (
        "🗓 <b>Тижневий розклад</b>\n"
        f"{format_week_type_footer(week_type, context='week')}\n\n"
        "<i>Показано понеділок — суботу. Посилання Meet/eLearn — у тексті, "
        "якщо вони задані для лекцій.</i>"
    )

    day_blocks: list[str] = []
    for day, schedule in week_schedule.items():
        day_text, _ = format_schedule(
            schedule, DAY_NAMES[day], meet_inline_keyboard=False
        )
        day_blocks.append(day_text)

    divider = "\n\n——————————\n\n"
    text = header + divider + divider.join(day_blocks)

    if len(text) <= 4000:
        await message.answer(
            text,
            parse_mode=ParseMode.HTML,
            link_preview_options=_PREVIEW_OFF,
        )
        return
```

А.4.4. Фрагмент коду повернення до головного меню та перегляду профілю

У даному фрагменті наведено частину коду, яка відповідає за повернення користувача до головного меню та відображення інформації профілю.

```
from app.keyboards.main_menu import main_menu

@router.message(lambda m: m.text == ScheduleKB.BACK)
async def back_handler(message: Message) -> None:
    await message.answer(
        "🏠 <b>Головне меню</b>\n\n"
        "Повернув клавіатуру. Обери наступну дію 👇",
        parse_mode=ParseMode.HTML,
        reply_markup=main_menu,
    )


@router.message(lambda m: m.text == MainKB.PROFILE)
async def profile_handler(message: Message) -> None:
    user = get_user_by_telegram_id(message.from_user.id)

    if not user:
        await message.answer(
            "😕 <b>Профіль не знайдено</b>\n\n"
            "Натисни /start, щоб зберегти дані.",
            parse_mode=ParseMode.HTML,
        )
        return

    username = f"@{html.escape(user['username'])}" if user["username"] else "—"
    first_name = html.escape(user["first_name"] or "—")
    last_name = html.escape(user["last_name"] or "—")
    tid = html.escape(str(user["telegram_id"]))

    text = (
        "👤 <b>Твій профіль</b>\n\n"
        f"🆔 <b>Telegram ID</b>\n<code>{tid}</code>\n\n"
        f"✏️ <b>Ім’я</b>\n{first_name}\n\n"
        f"✏️ <b>Прізвище</b>\n{last_name}\n\n"
        f"📎 <b>Username</b>\n{username}"
    )

    await message.answer(text, parse_mode=ParseMode.HTML)
```

А.5. Фрагменти коду модуля роботи з вибірковими дисциплінами

У даному підпункті наведено фрагменти коду модуля app/handlers/electives.py, який реалізує перегляд, зміну та збереження вибіркових дисциплін користувача. Цей модуль є одним із ключових у системі, оскільки саме він забезпечує персоналізацію розкладу відповідно до індивідуального вибору студента.

А.5.1. Фрагмент коду перегляду вибіркових дисциплін

У даному фрагменті наведено частину коду, яка відповідає за перевірку наявності користувача в системі та відображення списку вже обраних вибіркових дисциплін.

```
import html

from aiogram import Router
from aiogram.enums import ParseMode
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.types import Message

from app.database.db import (
    add_user_elective_subject,
    clear_user_elective_subjects,
    get_all_elective_subjects,
    get_user_by_telegram_id,
    get_user_elective_subjects,
)
from app.ui_strings import MainKB

router = Router()


class ElectiveSelectionState(StatesGroup):
    waiting_for_subjects = State()


def _not_registered() -> str:
    return (
        "😕 <b>Користувача не знайдено</b>\n\n"
        "Натисни /start, щоб зареєструватися."
    )


@router.message(lambda m: m.text == MainKB.MY_ELECTIVES)
async def my_subjects_handler(message: Message) -> None:
    db_user = get_user_by_telegram_id(message.from_user.id)
```

```
if not db_user:
    await message.answer(_not_registered(), parse_mode=ParseMode.HTML)
    return

subjects = get_user_elective_subjects(db_user["id"])
if not subjects:
    await message.answer(
        "📚 <b>Вибіркові предмети</b>\n\n"
        "Поки що список порожній.\n\n"
        f"Натисни «{MainKB.EDIT_ELECTIVES}», щоб обрати дисципліни — "
        "тоді розклад підлаштується під тебе.",
        parse_mode=ParseMode.HTML,
    )
    return
rows = "\n".join(
    f"  <b>{idx + 1}.</b> {html.escape(subject['name'])}"
    for idx, subject in enumerate(subjects)
)
await message.answer(
    "📚 <b>Твої вибіркові</b>\n\n"
    f"{rows}\n\n"
    "<i>Ці предмети замінюють відповідні слоти в персональному розкладі.</i>",
    parse_mode=ParseMode.HTML,
)
```

А.5.2. Фрагмент коду запуску режиму редагування вибіркових дисциплін

У даному фрагменті наведено частину коду, яка формує список доступних вибіркових дисциплін та переводить користувача в режим введення нового набору предметів.

```
@router.message(lambda m: m.text == MainKB.EDIT_ELECTIVES)
async def change_subjects_handler(message: Message, state: FSMContext) -> None:
    subjects = get_all_elective_subjects()

    subject_text = "\n".join(
        f"  <code>{subject['id']}</code> · {html.escape(subject['name'])}"
        for subject in subjects
    )

    await message.answer(
        "✏️ <b>Зміна вибіркових</b>\n\n"
        "Надішли <b>номери предметів через кому</b> (як у списку нижче).\n\n"
        "📝 <b>Приклад:</b> <code>1, 3, 5</code>\n\n"
        "———\n"
        "<b>Доступні предмети</b>\n"
        f"{subject_text}\n\n"
        "💡 <i>Щоб скасувати введення, надішли /start.</i>",
        parse_mode=ParseMode.HTML,
    )

    await state.set_state(ElectiveSelectionState.waiting_for_subjects)
```

А.5.3. Фрагмент коду збереження вибіркових дисциплін користувача

У даному фрагменті наведено частину коду, яка перевіряє коректність введених даних, очищає попередній вибір користувача, зберігає нові дисципліни та формує підтверджувальне повідомлення.

```
@router.message(ElectiveSelectionState.waiting_for_subjects)
async def save_subjects_handler(message: Message, state: FSMContext) -> None:
    db_user = get_user_by_telegram_id(message.from_user.id)

    if not db_user:
        await message.answer(_not_registered(), parse_mode=ParseMode.HTML)
        await state.clear()
        return

    raw_input = (message.text or "").replace(" ", "")
    parts = raw_input.split(",")

    try:
        subject_ids = [int(part) for part in parts if part]
    except ValueError:
        await message.answer(
            "⚠️ <b>Не зовсім зрозумів формат</b>\n\n"
            "Потрібні лише цифри через кому, наприклад:\n<code>1,3,5</code>",
            parse_mode=ParseMode.HTML,
        )
        return

    all_subjects = get_all_elective_subjects()
    valid_ids = {subject["id"] for subject in all_subjects}

    if not subject_ids:
        await message.answer(
            "🤔 <b>Список порожній</b>\n\n"
            "Вкажи хоча б один номер предмета.",
            parse_mode=ParseMode.HTML,
        )
        return

    invalid_ids = [sid for sid in subject_ids if sid not in valid_ids]
    if invalid_ids:
        bad = ", ".join(html.escape(str(i)) for i in invalid_ids)
        await message.answer(
            f"⚠️ <b>Невідомі номери:</b> {bad}\n\n"
            "Перевір список і спробуй ще раз.",
            parse_mode=ParseMode.HTML,
        )
        return

    clear_user_elective_subjects(db_user["id"])

    unique_subject_ids = list(dict.fromkeys(subject_ids))
    for subject_id in unique_subject_ids:
        add_user_elective_subject(db_user["id"], subject_id)
```

```
selected_subjects = [
    subject["name"]
    for subject in all_subjects
    if subject["id"] in unique_subject_ids
]
lines = "\n".join(f"  ✓ {html.escape(name)}" for name in selected_subjects)

await message.answer(
    "✅ <b>Готово!</b>\n\n"
    "Оновив твій список вибіркових:\n"
    f"{lines}\n\n"
    "🗓 Заглянь у «Розклад» — там уже враховані ці предмети.",
    parse_mode=ParseMode.HTML,
)

await state.clear()
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Ляховчук Владислав Вікторович , здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему

« Програмне забезпечення для автоматизації розкладу занять студентів на базі Telegram-бота » є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [✓] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [✓] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ **Владислав ЛЯХОВЧУК**

(підпис)